1983

Multiplexing the Ethernet interface among VAX/VMS users

Sakellaropoulos, Antonios K.; Kidoniefs, Ioannis K.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

MULTIPLEXING THE ETHERNET INTERFACE
AMONG VAX/VMS USERS

by

Antonios K. Sakellaropoulos
and
Ioannis K. Kidoniefs

December 1983

Thesis Advisor: Uno R. Kodres

Approved for public release; distribution unlimited

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Multiplexing the Ethernet Interface Among VAX/VMS Users		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis December 1983
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Antonios K. Sakellaropoulos Ioannis K. Kidoniefs		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943		12. REPORT DATE December, 1983
		13. NUMBER OF PAGES 154
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Ethermult

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This thesis focuses on the multiplexing of Ethernet interface among VAX 11/780 users.
Since there is only one channel that connects VAX 11/780 system to Ethernet Local Area Network, multiplexing of the NI1010 Unibus Ethernet Communication Controller is necessary in order to service multiple VAX users concurrently via Ethernet.
Described herein is a Time Division Multiplexing (Continued)

DD FORM 1473 1 JAN 73 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

ABSTRACT (Continued)

of NI1010 controller, which can serve up to nine (9) separate VAX users. The developed software enables those users to communicate in any combination with one or more computer systems, which are connected to Ethernet as well.
Two Microcomputer Development Systems (MDS) and VAX/VMS system were used for the implementation and testing of the project.
The software is designed in such a way that those MDS's act very much like virtual VAX/VMS terminals.
The whole system can easily be expanded to serve more than nine users.

Approved for public release; distribution unlimited.

Multiplexing the Ethernet Interface Among VAX/VMS Users

by

Antonios K. Sakellaropoulos*
Major, Hellenic Air Force
B.S., Hellenic Air Force Academy, 1972

and

Ioannis K. Kidoniefs**
Lieutenant, Hellenic Navy
B.S., Hellenic Naval Academy, 1975

Submitted in partial fulfillment of the requirements for the degrees of

MASTER OF SCIENCE IN COMPUTER SCIENCE*
MASTER OF SCIENCE IN ENGINEERING SCIENCE**

from the

NAVAL POSTGRADUATE SCHOOL
December 1983

ABSTRACT

This thesis focuses on the multiplexing of Ethernet interface among VAX 11/780 users.

Since there is only one channel that connects VAX 11/780 system to Ethernet Local Area Network, multiplexing of the NI1010 Unibus Ethernet Communication Controller is necessary in order to service multiple VAX users concurrently via Ethernet.

Described herein is a Time Division Multiplexing Of NI1010 controller, which can serve up to nine (9) separate VAX users. The developed software enables those users to communicate in any combination with one or more computer systems, which are connected to Ethernet as well.

Two Microcomputer Development Systems (MDS) and VAX/VMS system were used for the implementation and testing of the project. The software is designed in such a way that those MDS's act very much like virtual VAX/VMS terminals.

The whole system can easily be expanded to serve more than nine users.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

A. DISCLAIMER

Many terms used in this thesis are registered trademarks of commercial products. Rather than attempt to cite each individual occurence of a trademark, all registered trademarks appearing in this thesis will be listed below, followed by the firm holding the trademark:

Ethernet	Xerox Corporation
Unibus, VAX, VMS	Digital Equipment Corporation
MDS	Intel Corporation

B. CONVENTIONS

For reasons of convenience and readability, some of widely used terms in this thesis will be referred to hereafter as follows:

VAX/VMS	will imply the VAX 11/780 system operating under Virtual Memory System (VMS)
Ethernet	will stand for Ethernet Local Area Network
NI1010	will stand for NI1010 Unibus Ethernet Communication Controller (VAX-Ethernet) interface.

C. STRUCTURE OF THE THESIS

The rest of Chapter I gives a general development history of the project and describes the objectives of this thesis.

Chapter II provides the system layout as it currently exists. It describes how the VAX and the MDS's systems are connected via Ethernet and refers to the previous relevant work which has been performed by other students.

In Chapter III information which is very useful and sometimes necessary in order to understand this thesis, is provided. These pieces of information were found in a variety of references and it would be very painful for the reader to try to locate them on his own.

The material up to and including Chapter III constitutes the background for the understanding of this thesis. In Chapter IV however, the real work starts with the high level design of the Ethernet multiplexing. In this chapter the design concept is described, and justification for the decisions that were made is given.

Chapter V contains the detailed design and implementation of the project. In reality Chapter V constitutes the documentation of the developed software. In some instances, things that have been mentioned in previous parts of this thesis are repeated, when it was thought that they are necessary for the thorough understanding of the program.

Chapter VI contains the conclusion of this work.

This thesis is also supported by several appendices.

Appendix A provides a description of VAX/VMS system services and the Run Time Library routines which were used in the programs, along with information on their use.

Appendix B provides general information about the Ethernet local area network.

Appendix C gives a short description of NI1010 controller which constitutes the VAX - Ethernet interface.

Appendix D contains the User's Manual for the multiplexing of VAX/VMS-Ethernet interface.

Appendices E and F provide the two programs which compose this project (see Chap. IV B).

Appendix G includes the modified programs of a previous thesis by Mark Stotzer (see Chap. II B 1) with a brief explanation of the changes.

Appendix H provides a high level design of a virtual terminal network, along with some hints which may be useful to the person who will work in this area.

Finally in Appendix I is given the program "LOGER" which creates a detached process to run the image "LOGINOUT.EXE". More information about this effort can be found in Chapter 6.

D. THESIS OBJECTIVES

When this thesis started, the objective was to create a network of virtual terminals for the VAX/VMS system. More specifically, two MDS microcomputers which were connected to the Ethernet local area network, had to be made to act like virtual reminals of VAX/VMS which is also connected to the Ethernet.

After most of the work was done and the greatest part of the project had been designed and implemented, all that was left was the invocation of the Loginout procedure [Ref. 1] from an MDS terminal. But at that point it was discovered that under the current design of the VAX/VMS Loginout procedure, it is impossible to invoke this procedure by anything else except the physically connected VAX terminal. So the original goal was changed.

The new objective was the multiplexing of the Ethernet interface among VAX/VMS users i.e. NI1010 controller (see Appendix C). In other words, how it could be possible for various users of VAX/VMS to use concurrently the unique channel via which VAX is connected to Ethernet.

Since the original and the final objectives were in the same general direction, a large part of the original design

was maintained. The developed software finally provides a degree of virtual terminal service. The deviation from full virtual terminal service is that currently the Loginout procedure must be executed from a VAX terminal, and the program that is responsible for the execution of commands entered from an MDS terminal must also run from a VAX/VMS terminal.

Since we need to occupy one real VAX terminal, in order to use an MDS as a virtual terminal, there is no practical usefulness in the virtual terminal service as it currently exists. However the Digital Equipment corporation is working on the modification of the Loginout procedure. As soon as this modified version of the Loginout procedure becomes available, this thesis can be relatively easily modified so that full virtual terminal service will be achieved.

In Appendix H a high level design of a virtual terminal network is provided, which may be useful to the person who will undertake this task when the Loginout problem will be eliminated.

II. THESIS BACKGROUND

Before the development of the present thesis, Stotzer [Ref. 2] and Netniyom [Ref. 3], had worked on the communication interface between MDS microcomputers and VAX/VMS. Specifically they worked on transferring single messages and files from VAX to MDS and vice versa.

A. SYSTEM LAYOUT

In order that communication between the VAX/VMS and the MDS systems to be achieved, both of them were connected to Ethernet local area network. Figure 2.1 depicts how the MDS's and the VAX are currently connected. This configuration existed on the fifth floor of the Spanagel Hall at Naval Postgraduate School, when this thesis was being developed.

One single density and one double density MDS are connected to the Ethernet. Each one of them is equipped with an NI3010 board [Ref. 4], which contains all the data communication controller logic required for interfacing those microcomputers to the Ethernet.

Similarly between the VAX and the Ethernet stands an NI1010A Unibus Ethernet communication controller [Ref. 5], which is also a single board that contains the required logic for interfacing the VAX to Ethernet.

Appendix B contains brief information about Ethernet. A short description of NI1010 controller is given in Appendix C.

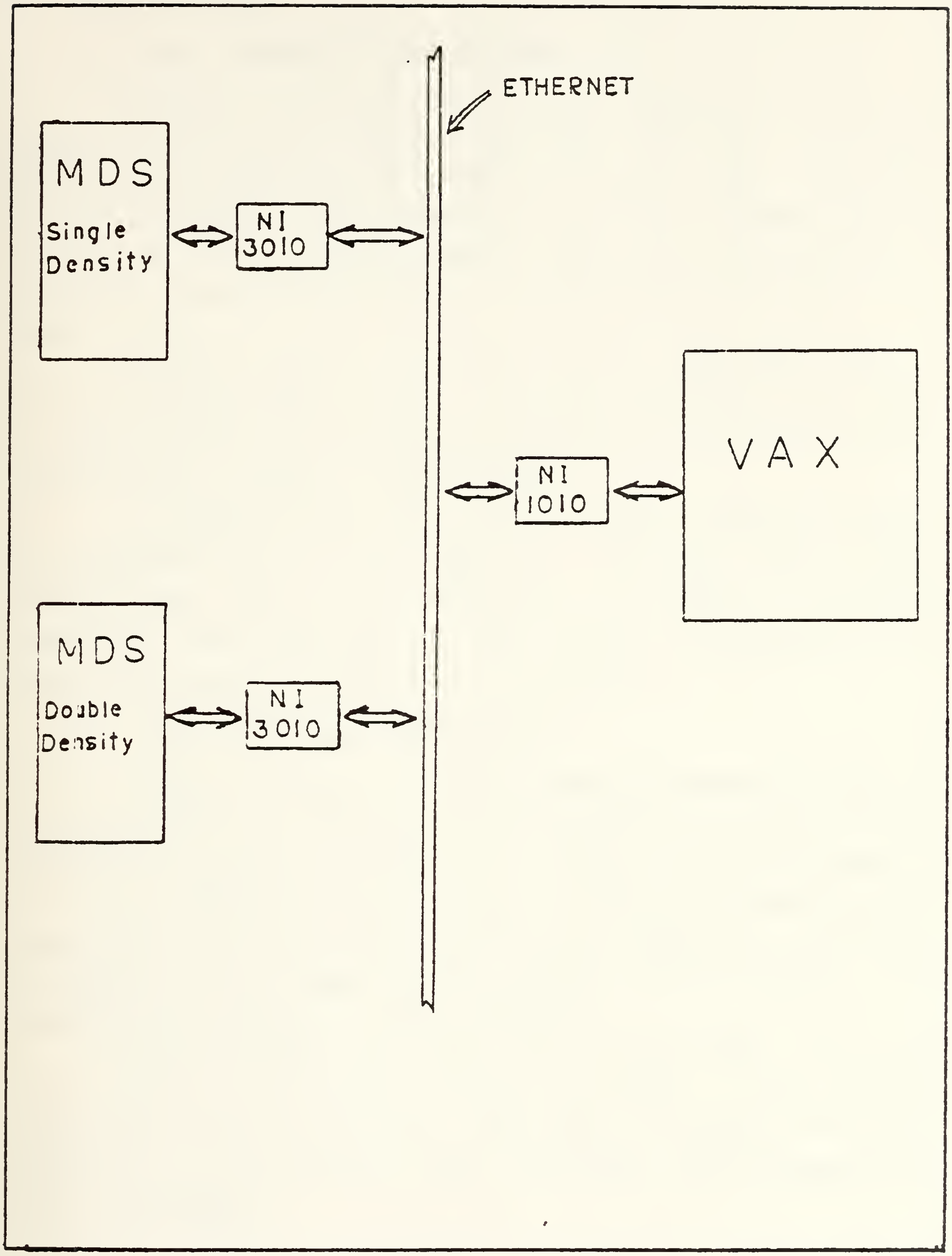

Figure 2.1 System Layout.

B. PREVIOUS WORK

1. MDS - Ethernet Communication

Stotzer had undertaken the part which deals with the software interface between the MDS and the Ethernet. Stotzer's programs were modified in order to meet the needs of a continuous ,uninterrupted, MDS-VAX communication, because in his original programs, after the transmission of a single message, one had to invoke the program again each time.

Since a part of this thesis was developed concurrently with Stotzer's thesis, several suggestions were made to him, and he redesigned his programs so that they could be used for the purposes of this thesis.

In spite of the new effort, after Stotzer's thesis was completed and he had departed, very few but crucial changes were made to his software, in order it to serve completely the objectives of this thesis. In Appendix G the modified Stotzer's programs are provided.

2. VAX - Ethernet Communication

Netniyom worked on the interface between the VAX and the Ethernet. His programs made possible the receiving and sending of messages and files from and to the NI1010 board.

According to his design, a process is continuously monitoring the NI1010 and as soon as it receives a message, it sends an aknowledge to the source of the message, and displays this message on the terminal.

This process can serve only one VAX user at a time and no other process can establish access to the NI1010 unless the previous one has terminated, freeing the unique channel of communication with the NI1010 and consequently with the Ethernet.

In contrast to Stotzer's programs which were used almost unchanged, Netniyom's software was not used. However, his work was studied most carefully by the writers of this thesis, who obtained the initial knowledge of the VAX-Ethernet communication from his document.

III. RELEVANT INFORMATION

Before proceeding with the design of the multiplexing system of the Ethernet interface among VAX - 11 users, it would be more convenient for the reader if some relevant background information was provided. In this way the reader will save the trouble to find this information on his own and thus discrepancies in assumptions will be avoided.

A. MULTIPLEXING

In general terms the word multiplexing refers to the use of a single facility to handle concurrently several similar but separate operations. The main use of multiplexing however, is in the field of data communications, where it is used for the transmission of several lower speed data streams, over a single higher speed line.

In the context of this project, multiplexing of the Ethernet interface implies a scheme under which many VAX users share the unique channel of communication between VAX and Ethernet.

Multiplexing is divided into two basic categories: Frequency Division Multiplexing (FDM) and Time Division Multiplexing (TDM).

In FDM the frequency spectrum is divided up among the logical channels, with each user having exclusive possession of his frequency band.

In TDM the users take turns (in a round robin fashion), each one periodically getting the entire bandwidth for a short burst at a time.

The latter scheme was used for the multiplexing of Ethernet interface among VAX users. Each user who requests

service via Ethernet, makes exclusive use of the VAX - Ethernet commmunication channel for a short period of time. Then another user takes control of the channel for a while, and so on. When all the users have used the channel, control comes back to the first user and a new cycle begins.

B. PROCESS

1. Process Definition

Process is the fundamental program unit in VAX/VMS which is selected by the scheduler for execution [Ref. 1]. A process is automatically created for each user when he logs on. The user runs programs, one at a time, in his process. Only one program can run at a time in any process. A process is identified by a process ID or PID.

A process is fully described by hardware context, software context and virtual address space description.

a. Hardware Context

The hardware context consists of copies of the general purpose registers, the four per process stack pointers, the program counter (PC), the processor status longword (PSL), and the process- specific processor registers including the memory management registers and the AST level register.

The hardware context resides in a data structure called the hardware process control block that is used primarily when a process is removed from or selected for execution.

b. Software Context

Software context consists of all the data required by various parts of the operating system, to make scheduling and other decisions about a process.

c. Virtual Address Space Description

The virtual address space of a process is divided into two regions:

- The program region (P0), which contains the image currently being executed.

- The control region (P1), which contains the information maintained by the system on behalf of the process. It also contains the user stack, which expands toward the lower-addressed end of the control region.

The following Figure 3.1 illustrates the layout

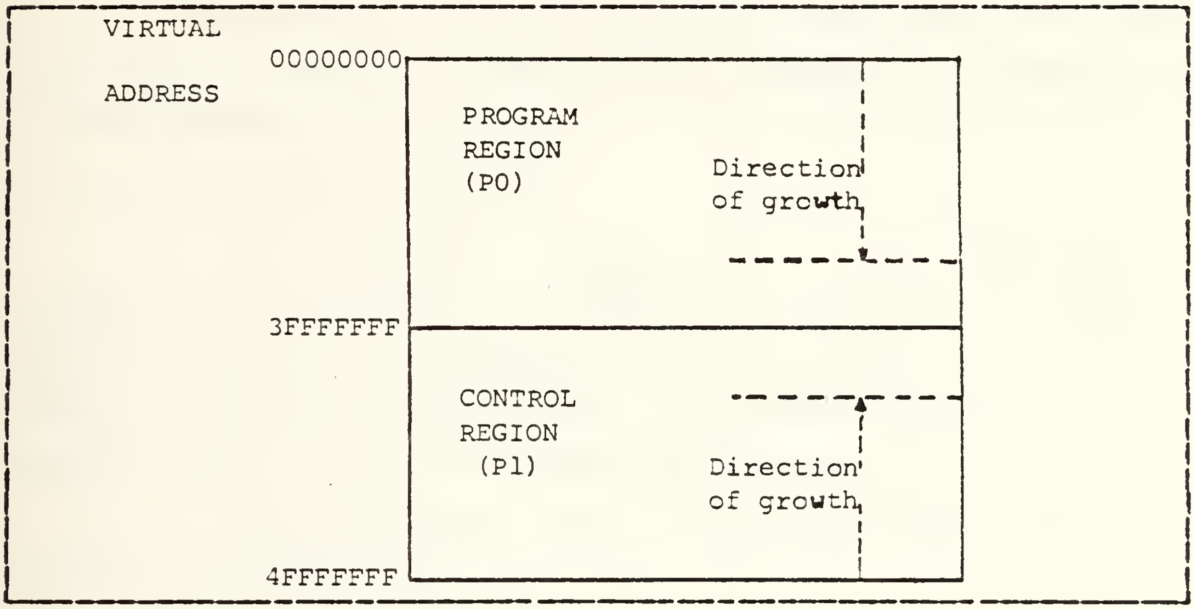

Figure 3.1 Layout of Process Virtual Address Space.

of a process's virtual memory.

The initial size of a process's virtual address space depends on the size of the image being executed. More information about virtual address space can be found in the Chapter 10 of VAX/VMS System Services Reference Manual [Ref. 6].

2. Job Definition

A process may create subprocesses, and those subprocesses may create new ones and so on. The collection of the creator process, all the subprocesses created by it, and all subprocesses created by its decendants, is called a job.

3. Image Defintion

The programs that execute in the context of a process are called images. Images usually reside in files that are produced by one of the VAX/VMS linkers.

4. Types of Processes

Processes are divided into two broad categories with diverse attributes. Those are the Detached Processes and the Subprocesses [Ref. 7].

a. Detached Process

The detached process is a fully independent process. The creation of a detached process is primarily a function performed by VAX/VMS at log in time. One can also create a detached process using the $CREPRC system service, provided that he has the DETACH priviledge.

The attributes of a detached process [Ref. 8] are the following:

- Has own resources
- Has own quotas
- May have a different user identification code (UIC) from creator
- Terminates independently of creator
- Detach priviledge required to create
- Cannot access creator's devices.

b. Subpocess

A subprocess receives a portion of its creator's resource quotas and must terminate before the creator. It can created by $CREPRC system service. When using $CREPRC service, if one does not specify the creation of a detached process, by default a subprocess is created.

In general, the attributes of a subprocess are:

- Shares creator's resources
- Shares creator's pooled quotas
- Has creator's UIC
- Must terminate before creator
- No priviledge required to create
- Subtracts from PRCLM quota
- Can access devices allocated to creator.

C. INTERPROCESS COMMUNICATION

During the entire design of the multiplexing system, major effort was placed in the efficiency and speed of the job execution. Instead of processes being executed sequentially, it was thought that they might be executed concurrently. In such a case the processes need to communicate among each other for reasons of synchronization and mutual exclusion. Thus the issue of interprocess communication is brought up.

VAX/VMS provides the necessary services (system routines), for achieving efficient inteprocess synchronization/communication.

The categories of these services are:

- Event Flag Services.

- Input/Output Services.

- Timer and Time Conversion Services.

Further on, a brief description of each category is given.

1. Event Flag Services

Event flags are the simplest form of interprocess communication and synchronization (within and among processes). They are a series of bits (one per flag), set or cleared by one process and examined by the same or different processes. For synchronization within a process, Local Event Flags are used. For communication/ synchronization among processes Common Event Flags are used. Common Event Flags can only be used to communicate among processes within the same User Identification Code (UIC) group. The event flag system services are used to establish, set, clear, examine and wait for event flags to be set.

Each event flag is associated with a unique decimal number. Event flag arguments in system service calls refer to these numbers. For example, if the flag 1 is specified in a call to the SYS$QIO system service, the event flag number 1 is set when the I/O operation completes. To allow manipulation of groups of event flags, the flags are ordered in clusters, with 32 flags in each cluster, numbered from right to left, corresponding to bits 0 through 31 in a longword. The clusters are also numbered from 0 to 3. The range of event flag numbers encompasses the flags in all clusters : Event flag 0 is the first flag in cluster 0, event flag 32 is the first flag in cluster 1 and so on.

The ranges of event flag numbers and the clusters to which they belong are summarized in the Table I.

The event flag system services used in the programs of this thesis are shown in the Table II. In the next section is given a more detailed description of those services. Information about their use can be found in the VAX/VMS System Services Reference Manual [Ref. 6].

It is important to note that there are some other ways also to set Event Flags. The following system services, some of which will be examined more closely later, accept an optional EFN argument, which specifies an event flag to be set when the operation is completed:

- Queue I/O Request ($QIO and $QIOW forms, $INPUT and$ OUTPUT macros).
- Queue Lock Request ($ENQ and $ENQW forms).
- Set Timer ($SETIMR).
- Update Section File on Disk ($UPDSEC).
- Get Job/Process Information ($GETJPI).
- Get System-Wide Information ($GETSYI).

TABLE I

Event Flag Clusters

CLUSTER NUMBER	FLAG NUMBER	DESCRIPTION	RESTRICTION
0	0 -31	Local Event Flags for general use by one process.	0 - avoid using. 24-31 - reserved
1	32-63	-"- -"-	
2	64-95	Assignable common event cluster.	Must be associated before use.
3	96-127	-"- -"-	-"- -"-

Note that each of the above system services clears the specified event flag before it begins the requested operation.

TABLE II

Summary of Event Flag Services

SERVICE NAME	FUNCTION(S)	DESCRIPTION
Associate Common Event Flag Cluster ($ASCEFC).	Creates a temporary or permanent common event flag cluster.	TQELM quota (for temporary) PRMCEB privilege (for permanent).
	Creates a common event flag cluster in memory shared by multi-processors.	SHMEM privilege
	Establishes association of a process with an existing common event flag cluster.	Group association
Set Event Flag ($SETEF)	Turns on an event flag (local or common)	None (for local) Group association (for common)
Clear Event Flag ($CLREF)	Turns off an event flag (local or common)	As for $SETEF.
Read Event Flags ($READEF)	Returns the status of all event event flags (local or common).	None (if local) Group association (if common)
Wait For Single Event Flag ($WAITFR)	Places the current process in a wait state pending the setting of an event flag in a local or in a common event flag cluster.	As in $SETEF

2. Input/Output Services

The I/O subsystem on VAX/VMS has a three-tiered organization. The top tier is VAX-11 Resource Management System (RMS) which provides access to files, unit record devices and certain foreign devices. All VAX-11 high level languages invoke VAX-11 RMS to perform I/O. Thus, VAX-11 FORTRAN READ and WRITE statements cause the compiler to generate calls to VAX-11 RMS.

The second tier is the Queue I/O system services which were used throughout the programs. They perform device dependent I/O and VAX-11 RMS generates calls to these services. A programmer uses the Queue I/O services when :

-- Accessing devices not supported by RMS (real time devices as the NI1010 board).

-- Performing I/O operations not supported by RMS (logical or physical I/O).

-- Performing I/O operations not supported by the language's interface to RMS. (Asynchronous I/O in FORTRAN).

The bottom tier is the device driver itself. The Queue I/O services act as the user interface to the device driver which is never directly accessed by the application programmer.

To maintain a level of device independence, VAX/VMS provides logical name services which allow programs using the Queue I/O services to be written without regard for the actual controller or unit number of the device. In fact, if VAX-11 RMS is used, instead of the Queue I/O services, the type of the device need not be considered by the programmer. Table III lists the most important Device-Dependent I/O services.

In Table IV are listed the two basic Mailbox and Message I/O services used in this thesis to provide the adequate communication among processes. After this overview of the most important I/O services, four of which, namely $ASSIGN, $QIO, $QIOW and $CREMBX appear in the programs of this thesis, it is considered useful to carry out a brief discussion of two basic features that VAX/VMS incorporates

TABLE III

MAILBOX SERVICES

Service name	Function(s)	Restriction(s)
Create Mailbox and Assign Channel ($CREMBX)	Creates a temporary or a permanent mailbox.	BYTLM quota. TMPMBX privilege (for temporary MBX). PRMMBX privilege (for permanent MBX).
Delete Mailbox ($DELMBX)	Marks a permanent mailbox for deletion.	PRMMBX privilege. Access mode.

in the above services: AST's and MAILBOXes. These features, constituting the backbone of the whole design, have provided the desired efficiency and facilitated the communication between processes.

a. Asynchronous Trap Services

Some system services allow a process to request that it be interrupted when a particular event occurs. Since the interrupt occurs (out of sequence) with respect to process execution, the interrupt mechanism is called an asynchronous system trap (AST). The trap provides a transfer of control to a user-specified procedure that handles the event.

The system services that use an AST mechanism accept as argument the address of an AST service routine

TABLE IV

SUMMARY OF I/O SERVICES

Service Name	Function(s)	Restrictions
Assign I/O Channel ($ASSIGN).	Establishes a path for an I/O request or network operations.	None (for I/O request). NETMBOX privilege (for network operations).
Deassign I/O Channel ($DASSIGN).	Releases linkage for an I/O path. Releases a path from the network.	Access mode
Queue I/O request ($QIO)	Initiates an input or output operation.	Access mode
Queue I/O request and Wait for event flag ($QIOW).	Initiates an input or output operation and causes the process to wait until it is completed before continuing execution.	Access mode
Allocate Device ($ALLOC).	Reserves a device for exclusive use by a process and its subprocesses.	None
Deallocate Device ($DALLOC).	Relinquishes exclusive use of a device.	Access mode

that is, a routine to be given control when the event occurs. These services are:

- -- Declare AST ($DCLAST)
- -- Enqueue Lock Request ($ENQ)
- -- Get Device/Volume Information ($GETDVI)
- -- Get Job/Process Information ($GETJPI)
- -- Get System Wide Information ($GETSYI)

-- Queue I/O Request ($QIO)

-- Set Timer ($SETIMR)

-- Set Power Recovery AST ($SETPRA)

-- Update Section File On Disk ($UPDSEC)

Of the above, the $QIO and the $SETIMR services used in the programs, include in their arguments ASTs.

ASTs are queued for a process by access mode. An AST for a more privileged access mode always takes precedence over one for a less privileged access mode; that is, an AST will interrupt any AST service routine executing at a less privileged mode; however, the process can receive ASTs from more privileged access modes (for example a Kernel-mode AST at I/O completion).

Figure 3.2 shows a program interrupted by a user-mode AST, and the user-mode AST service routine interrupted by a Kernel-mode AST :

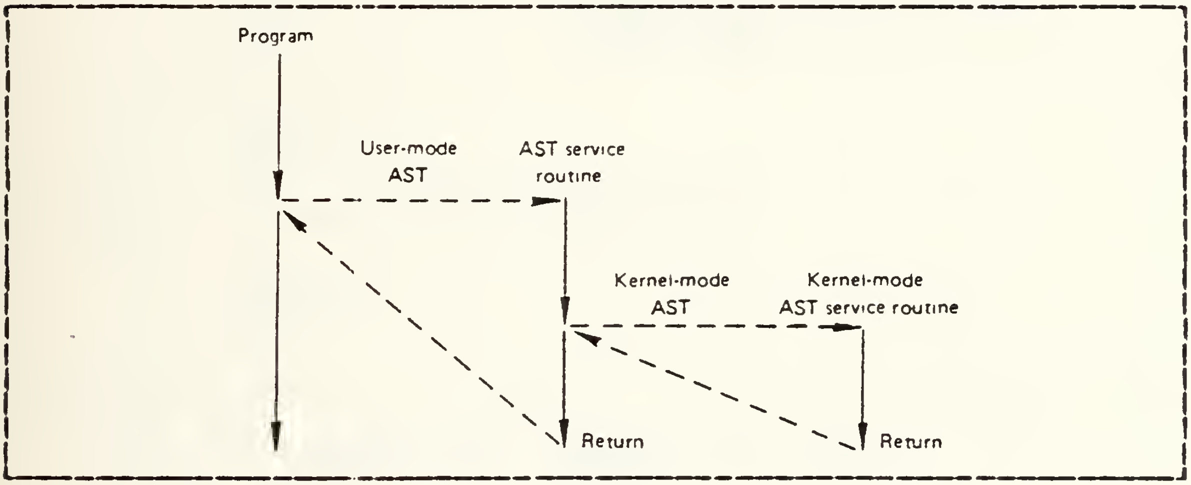

Figure 3.2 Access Modes and AST Delivery.

Some examples are given below, which may clarify the way ASTs work.

__Example with $SETIMR AST__

In the Set Timer ($SETIMR) system service, one can specify the address of a routine (subroutine or function) in the main program to be executed when a time interval expires or when a particular time of day is reached. The service schedules the execution of the routine and returns. Up to this point the sequence of the program execution has not been changed. Now when the requested timer event occurs, the system "delivers" an AST __by interrupting the process and calling the specified routine__.

```
          subroutine rec(MRpacket)
                      .
                      .
          implicit    integer*4 (a-z)
#1.       external    dummy
                      .
                      .
          istat = sys$qio (%val(1),%val(nichan),
#2.       1                 %val(io$_readlblk,iosb,
          2                 dummy,MRpacket,MRpacket,
          3                 %val(1522),,,,)
                      .
                      .
          return
          end
                      .
                      .
#3.       subroutine dummy(MRpacket)
                      .
                      .
          return
          end
```

Figure 3.3 Example of an AST.

__Example with $QIO AST__

In this example, the $QIO service is called. You can now specify not only the address of a routine but also the parameter to be passed to this routine. Figure 3.3 taken from the program "Ethermult" shows how an AST can be delivered.

Notes on Figure 3.3:

#1. The AST subroutine should be declared as "external".

#2. The AST subroutine name (address) and its parameter are among the arguments of the $QIO service, at the proper positions. The service is executed in the normal sequence of the program, sets the receive mode and returns. The program continues executing until the I/O is completed, i.e. until a packet is received. When this happens, the AST is "delivered", the program is interrupted, and control is transferred at the subroutine "dummy" which is executed. When control returns, the program continues executing from the point of interruption.

More about ASTs can be found in :

-- VAX/VMS Real Time User's Guide [Ref. 9].

-- VAX/VMS System Services Reference Manual [Ref. 6].

b. Mailboxes

Mailboxes are synchronous (mainly) virtual devices which may be used to transfer information among cooperating processes. The amount of information transferred via mailboxes is normally less than 512 bytes. Actual data transfer is accomplished by using VAX/11 RMS or I/O services.

When the Create Mailbox and Assign Channel ($CREMBX) service creates a mailbox, it also assigns a channel to it for use by the creating process. Other processes can then assign channels to the mailbox using either the $CREMBX or the $ASSIGN system services.

The $CREMBX system service creates the mailbox. It identifies the mailbox by a user-specified logical name and assigns it an equivalence name. This name is a physical device name in the format MBAn: where n is a unit number.

Mailboxes are either temporary or permanent. The user privileges TMPMBX and PRMMBX are required to create temporary or permanent mailboxes respectively. The temporary mailbox is deleted as soon as the image that created it, ceases to exist.

THe $QIO (or $QIOW) system service is used to perform I/O to the mailbox. In a usual sequence of operations involving a mailbox, $CREMBX, $QIO and $QIOW system services are used. First, the $CREMBX creates the mailbox and assigns a channel to it and then the $QIO (or $QIOW) reads or writes to it. Other processes may have access to it by means of a distinct channel assigned to the process and can use also $QIO to read or write to it. In general, mailboxes are one of the simplest facilities to use that VAX/VMS provides for interprocess communication.

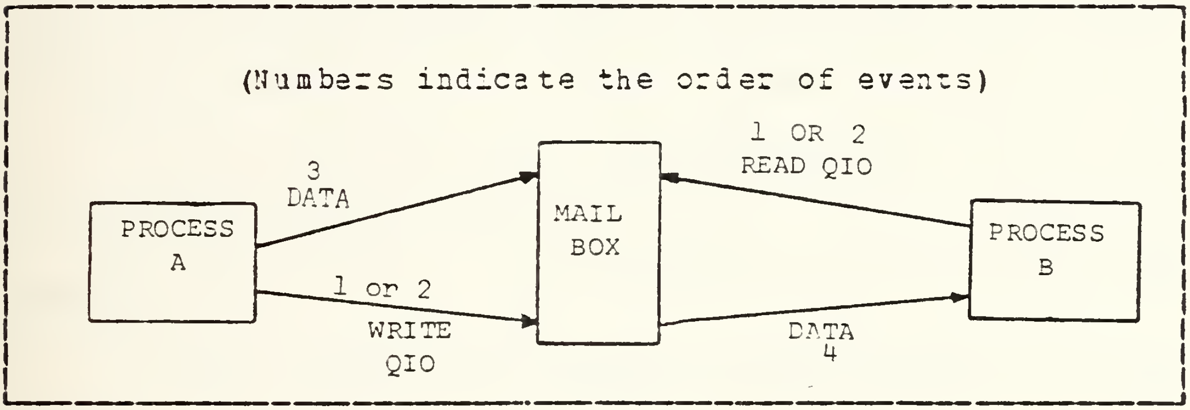

Figure 3.4 Two processes accessing a mailbox.

Figure 3.4 gives a schematic interpretation of the communication between processes via a mailbox.

More information about mailboxes can be found in:

-- VAX-11 Fortran User's Guide [Ref. 10].

-- VAX/VMS Real Time User's Guide [Ref. 9].

-- VAX/VMS System Services Reference Manual [Ref. 6].

3. <u>Timer and Time Conversion Services</u>

Many applications require the scheduling of program activities based on clock time. Under VAX/VMS, a process may schedule events for a specified time interval. Time services can do the following :

(1). Schedule the setting of an event flag, or queue of an asynchronous system trap (AST) for the current process, or cancel a pending request that has not yet been honored.

(2). Schedule a wake-up request for a hibernating process, or cancel a pending wake-up request that has not yet been honored.

(3). Set or recalibrate the current system time, if the caller has the proper user privileges.

The timer services require the user to specify the time in a unique 64-bit format. To work with the time in different formats, one can use time conversion services to :

(1). Obtain the current date and time in an ASCII string or in system format.

(2). Convert an ASCII string into the system time format.

(3). Convert the time from system format to integer values.

The following table lists the timer and time conversion services.

TABLE V

Basic Timer and Time Conversion Services

Service Name	Function(S)	Restriction
Get Time ($GETTIM).	Returns the date and time in system format.	None
Convert Binary Time to Numeric Time ($NUMTIM).	Converts a date and time from system format to numeric integer values.	None
Convert Binary Time to ASCII string ($ASCTIM).	Converts a date and time from system format to an ASCII string.	None.
Convert ASCII string to Binary ($BINTIM).	Convert a date and time in an ASCII string to the system format.	None.
Set Timer ($SETIMR)	Requests setting of an event flag or queuing of an AST based on absolute or delta time value.	TQELM quota.
Cancel Timer request ($CANTIM).	Cancels previously issued timer requests.	Access mode.

IV. DESIGN CONCEPT

A. GENERAL

1. Evolution Of The Design

The design for multiplexing the NI1010, the Ethernet interface device, was changed several times, as a part of it was implemented and experience was obtained. One major problem with this work was that the writers of this thesis had not worked in the past with VAX/VMS and sometimes it couldn't be determined in advance what could be done and what couldn't. Much experimentation took place and the original design was modified when necessary.

After the difficulties of these details were overcome, the first working version of the project was completed. However this program was not sufficiently fast, because the multiplexing was taking place at the level of the users. That is, when a user was being served, the program was staying with him until this user was fully served. For example when a user had to get a file from VAX/VMS, no other user would be served until the entire file of the current user was displayed on the MDS screen. If the file was large enough, considerable amount of time was spent, with the entire system devoted to one user, and that made the system very inefficient. Thus, it was decided that multiplexing occur at a lower level and instead of multiplexing users, the new design was multiplexing frames of the users.

2. Language Selection

The language that was selected for the implementation of the project was VAX-11 FORTRAN. The reason for this

choice was that FORTRAN is very well suported by VAX/VMS, and there are very powerful system routines and other facilities which make the job of the programmer much easier. Again it took some time to become familiar with this language which was mostly unknown to the designers.

3. Frame Size

Recall that information is transferred via Ethernet (see Appendix A) in groups of bytes. One such group is called a frame. Data in an Ethernet frame must be greater than 46 bytes and less or equal to 1500 bytes.

One decision that had to be made was about the size of the information frame to be transmitted each time.

Although testing and analysis was not very extensive, it was obvious that a frame size of 1500 was the most efficient, considering the average length of the responses to the various commands entered from an MDS terminal. This is reasonable, because after the transmission of each frame, the transmitter should receive an acknowledge from the receiver, and other secondary operations must therfore be executed. So, the smaller the frame size, the more frequently those parasite operations are executed. Also many subroutine calls are avoided. All of these operations are time consuming and reduce the efficiency of the program. For the above reason the largest frame appears to be the most efficient.

The only case where the large frame is at a disadvantage is when the message is very short. For example, if we have a message 80 bytes long, 1500 bytes still have to be transmitted but the 1420 of them will be empty. This situation may frequently occur with the last frame of a file. However since the transfer rate of Ethernet is so high (10 megabits per second) this is not very costly as far as time is concerned. So in overall, the highest frame size is still the most efficient among fixed size frames.

Variable length frames could be used in the program, to make it more efficient. However Stotzer in his program provides only for a fixed frame length of 1500 bytes. Since the frame length in both programs must be the same it was decided that this frame length be maintained.

B. HIGH LEVEL DESIGN

As was mentioned before, the Loginout VAX procedure cannot be invoked from a terminal other than an original VAX terminal. Because of this, a way had to be found to have commands entered from an MDS terminal to be executed in the VAX environment. The solution to this problem is the system routine "Spawn" (see Chap. 3), which takes as input, among other things, a VMS command and executes it. Unfortunately the only command that "Spawn" cannot execute is the Loginout procedure.

It was obvious from the first steps of the design phase that each user should have a program to execute the commands that are addressed to him. Since more than one user could require use of the Ethernet interface concurrently, there had to be another program which will coordinate the users, control the message traffic and assure the interleaving of the possesion of the communication channel.

In the following paragraphs, a high level design of the coordinator program and the user's program is provided. To avoid confusion, from now on the coordinator program will be called "ethermult" and the user's program will be called "usermult".

1. Program "Ethermult"

The program which will perform the Ethernet multiplexing should be able to perform the following operations:

First, it should be able to establish a channel of communication with the NI1010 board and sense any message that arrives there.

Second, it must check the validity of any incoming messages and determine the user to which a valid message is addressed,and then send the message to the appropriate user.

Third,it must check which user has something to send to the NI1010 and do it in an efficient and fast way, according to specified priorities.

It was thought that a lot of time would be saved if those three tasks are executed concurrently and not one after the other.

a. Use of Common Event Flags in "Ethermult"

In order that this program have control over the various user's programs, and know the status of the network at any moment (i.e. how many users are currently logged in, what user numbers they have etc.) a cluster of common event flags was used. The cluster which was used contains a total of 32 flags numbered from 64 to 95. Any program in the VAX/VMS environment may access this cluster and read, set, or reset any of those flags.

Flags #64 through #72 correspond to 9 users and they represent user numbers from 1 to 9. when any one of those flags is raised, this means that the user it represents, has logged in succesfully onto the network. A user number flag is set and reset by the "Usermult" program, as will be seen later.

You may have noticed already that the system can accept and serve up to nine users. However, this number may be easily expanded to a large number of users by assigning more flags to user numbers, and changing the dimensions of some data structures that are used in the program.

The limitation in user numbers is imposed due to the number of common event flags available in VAX/VMS, namely 96 flags arranged in three clusters of 32 flags each. Under the current design, the system can be expanded to serve a maximum of 47 users.

The flags from #73 through #81 are used to indicate whether the user who corresponds to each of them has an answer ready to send back to the MDS which has sent a command to him.

Figure 4.1 illustrates the use of common event flags for interprocess communication.

b. Use of Mailboxes in "Ethermult"

The communication of the "Ethermult" program with the "Usermult" programs is achieved through mailboxes. A received message is identified and put in the mailbox of the user to whom it is addressed. The message is then picked up by the "Usermult" for further elaboration.

Each user has his own mailbox which stays there as long as the corresponding "Usermult" is running. When a "Usermult" program terminates, the corresponding mailbox disappears. Figure 4.2 depicts the interprocess communication by means of mailboxes.

c. Use of AST's in "Ethermult"

Recall that an Asynchronous System Trap is nothing else but an interrupt which transfers control at a specified place of the program when a specific event occurs. "Ethermult" uses "Sys$qio" system service to continuously listen to the NI1010 board. This system service sets the "Ethermult" into a receive mode in which the program does not wait for the event to occur, but continues normal execution. As soon as a message arrives at the NI1010 board, an interrupt is triggered and control of the program is

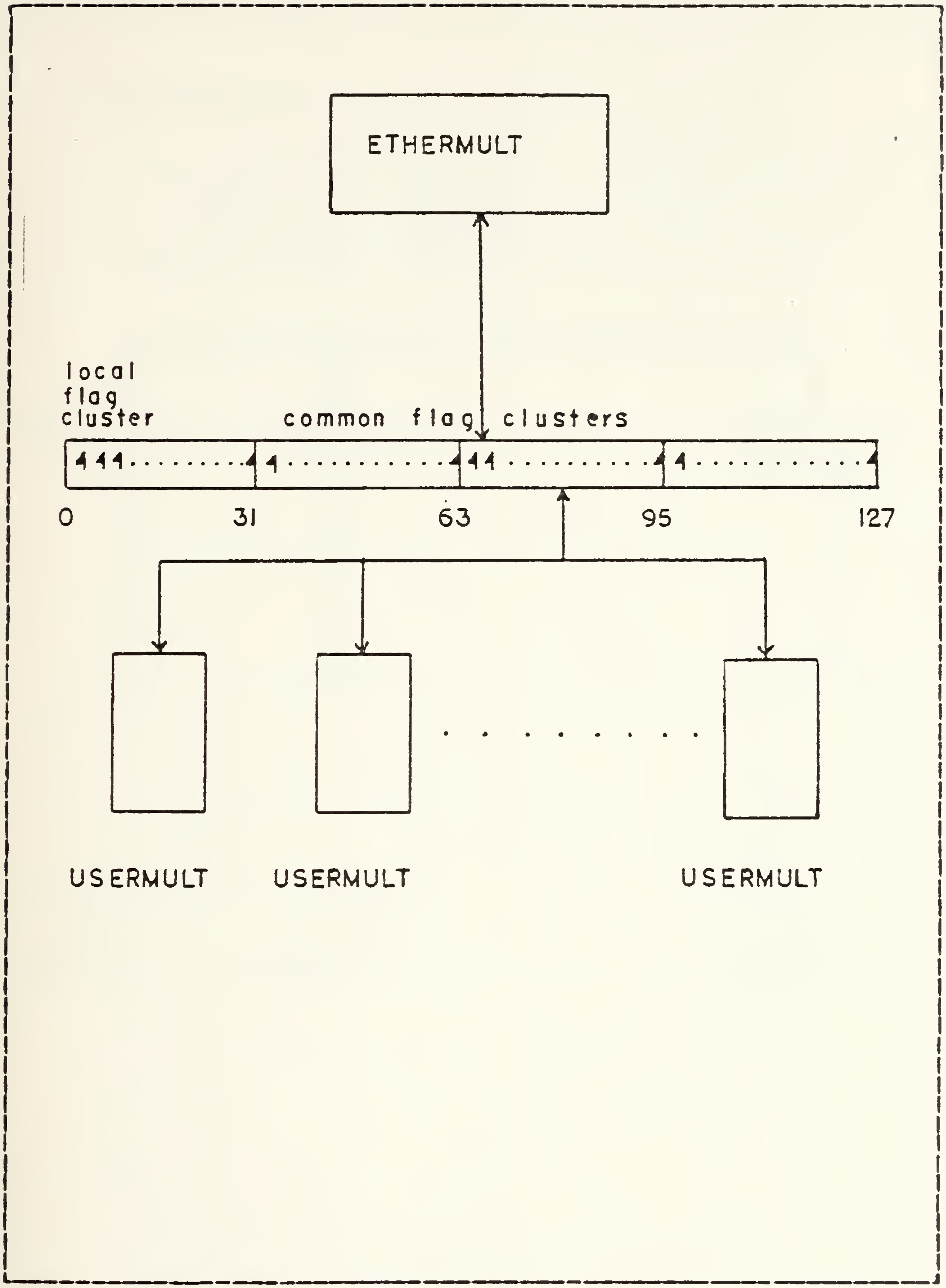

Figure 4.1 Use of Event Flags.

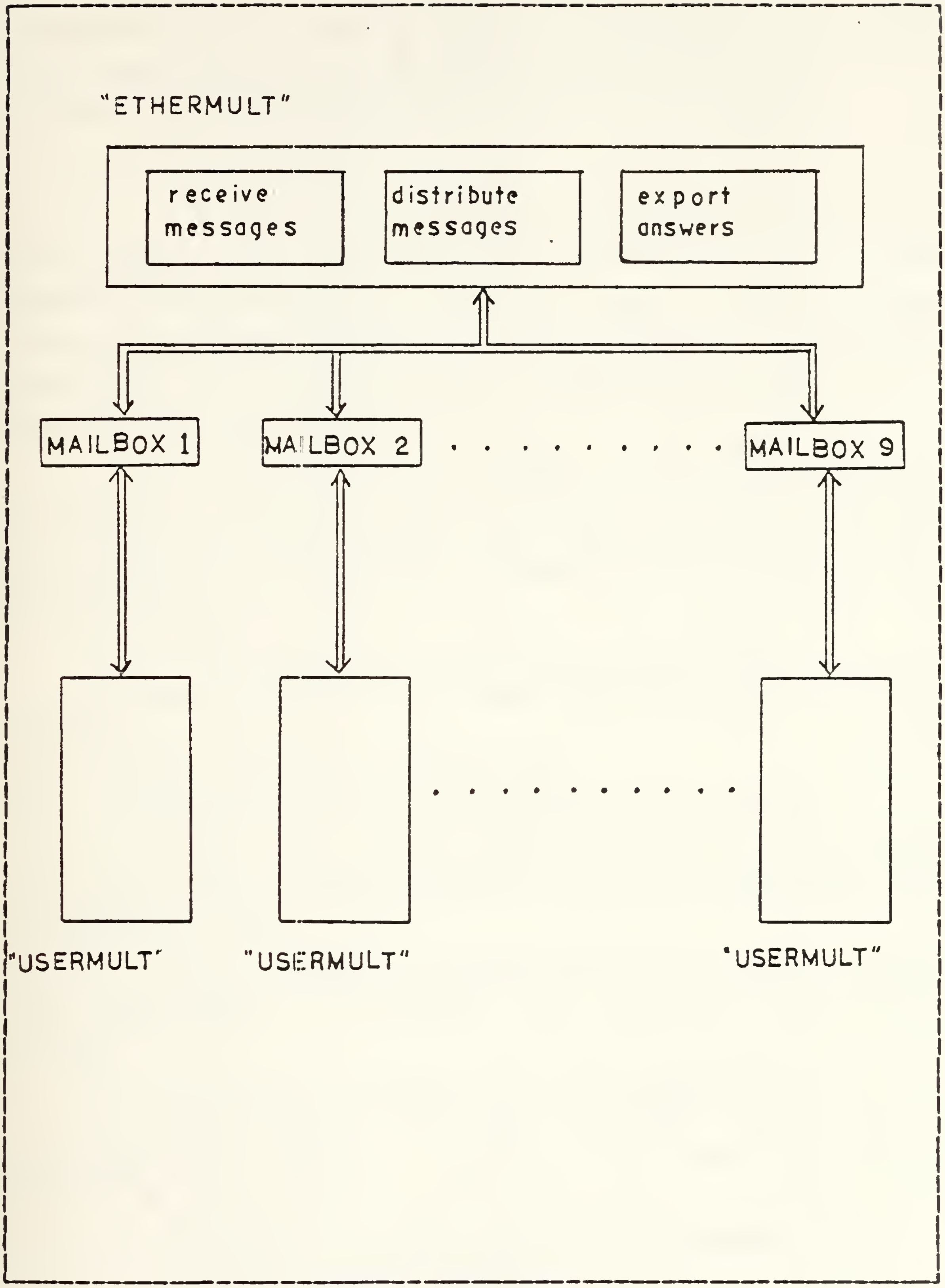

Figure 4.2 Process Communication Via Mailboxes.

transferred to a location which can be prespecified as an input parameter of the "Sys$qio". In this way there is no overhead due to the need of listening to the network.

d. "Ethermult" Algorithm

A high level algorithm of program "Ethermult" is given below. This algorithm does not go deep into details, because its purpose is to help the reader realize quickly what are the main operations of the program. A detailed description of the program and explanation of the code is given in the next chapter.

At this point it should be mentioned that the program uses a message queue in which the incoming messages are stored, in a FIFO order. Also there exists a user information table which contains information pertaining to the current users.

The high level algorithm is as follows:

1. Make accessible the common event flag cluster to the program.
2. Clear flags #64 through #84
3. Establish communication channel with NI1010
4. Set the "ear" of the system to listen to the network and specify where program control should be transferred when a message arrives.
5. When a message arrives, put it in the message queue, send acknowledge to the source, and reset the "ear" of the system.
6. Take the first message from the message queue.
7. Check whether the user to whom this message is addressed, is currently authorized to use the system.
8. If he is not:
    - send a caution message to the message source
    - disregard the message
    - rearrange the message queue

    Else

- rearrange the message queue
- put the message into the appropriate mail box
- if this is the first message for a VAX user:
  - update the user information table.

9. Check which users have a ready answer and send each answer to the proper destination, using a round robin scheme. According to this scheme each user is serviced for as much time as it is required for one frame to be sent, and acknowledge to be received. After this the next user is serviced.

10. Check the status of the network and update the user information table if necessary.

11. Repeat steps 5 to 11.

2. Program "Usermult"

a. Program Tasks

A user which uses an MDS terminal to access the VAX/VMS environment, is a virtual VAX/VMS user. As a VAX user he must be first authorized in order to use the system. This authorization is granted by the Loginout VAX/VMS procedure.

Since currently the Loginout procedure can be invoked only from a VAX terminal, there is no other way for a user but to use a VAX terminal in order to log on. In addition, because he intends to use the Ethernet multiplexing as well, he has also to get a second authorization, to enter the network. So after a user has succesfully logged in the VAX/VMS, he must run a program which will enable him to enter the network. This program is the "Usermult" program.

"Usermult" checks the common event flags #73 through #81, that represent the users, and finds out which

of them are not already set. Then it associates them with user numbers and prompts the user to select one of those. If the user chooses a legal number, authorization to enter the system is given and the corresponding flag is raised to notify "Ethermult" about the new status. If the user made an invalid entry, the program gives him another chance.

Commands like DIR, TYPE, PRINT etc. which are entered from an MDS terminal and pertain to a specific user,must be executed and the results should be sent back to the MDS terminal.

The distribution of the messages which arrive at the NI1010 board is performed by program "Ethermult", which places each message into the appropriate mailbox, very much like a mailman does. But this requires that there exist such a mail box. So the first thing that "Usermult" should do, after the user has been accepted by the system, is to create this user's mailbox and establish a path of communication with it.

After the mailbox is created, "Usermult" should check constantly whether a message has arrived. As soon as a command arrives, the program calls "Spawn" system procedure which executes the command.

A more efficient way of checking the mailbox would be to set the "Usermult" in receive mode and then put it in hibernation while still in the receive mode. As soon as a message arrived, an interrupt would be triggered to wake up the process in order it to continue execution. This interrupt driven scheme is more efficient for multiuser systems because this way a time slice is saved. However, for reasons of simplicity, it was decided, to implement the first method.

The output of the routine "Spawn" is placed in a file which will be used by "Ethermult", to send the answer back to the calling MDS. Then the program checks the

mailbox again for the next message. The message traffic among VAX processes is illustrated in Figure 4.3.

b. User Mutual Exclusion

It may happen that more than one user requests access to the network at the same time. In such a case two users may select the same user number. If this happens many problems and confusion will be created. So there must be a way to avoid a situation like this.

The solution to the problem was again a common event flag. Specifically the very first thing "Usermult" does is to access the event flag cluster and check whether flag #84 is set. If it is so, it means that another user is requesting network service at this moment. Thus the program should wait and keep checking flag #84.

As soon as this flag is reset, the requesting program raises it, so that no other user will interfere until a user number is obtained.

c. Protection

Now another issue is brought up. If no protection is provided, a user who has control of event flag #84, is enabled to prevent any other user from initiating a log in procedure. The solution which was adopted for this problem was the use of a watchdog timer.

A timer which is set at 10 seconds is energized when the user is prompted to make his selection, and a message on the screen notifies the user that he has 10 seconds to select a user number. If he does not make a selection within 10 seconds, flag #84 will be reset, and his program will terminate.

The program also resets flag #84 and terminates execution when the user makes two consecutive wrong selections. This way he cannot prevent other users from logging

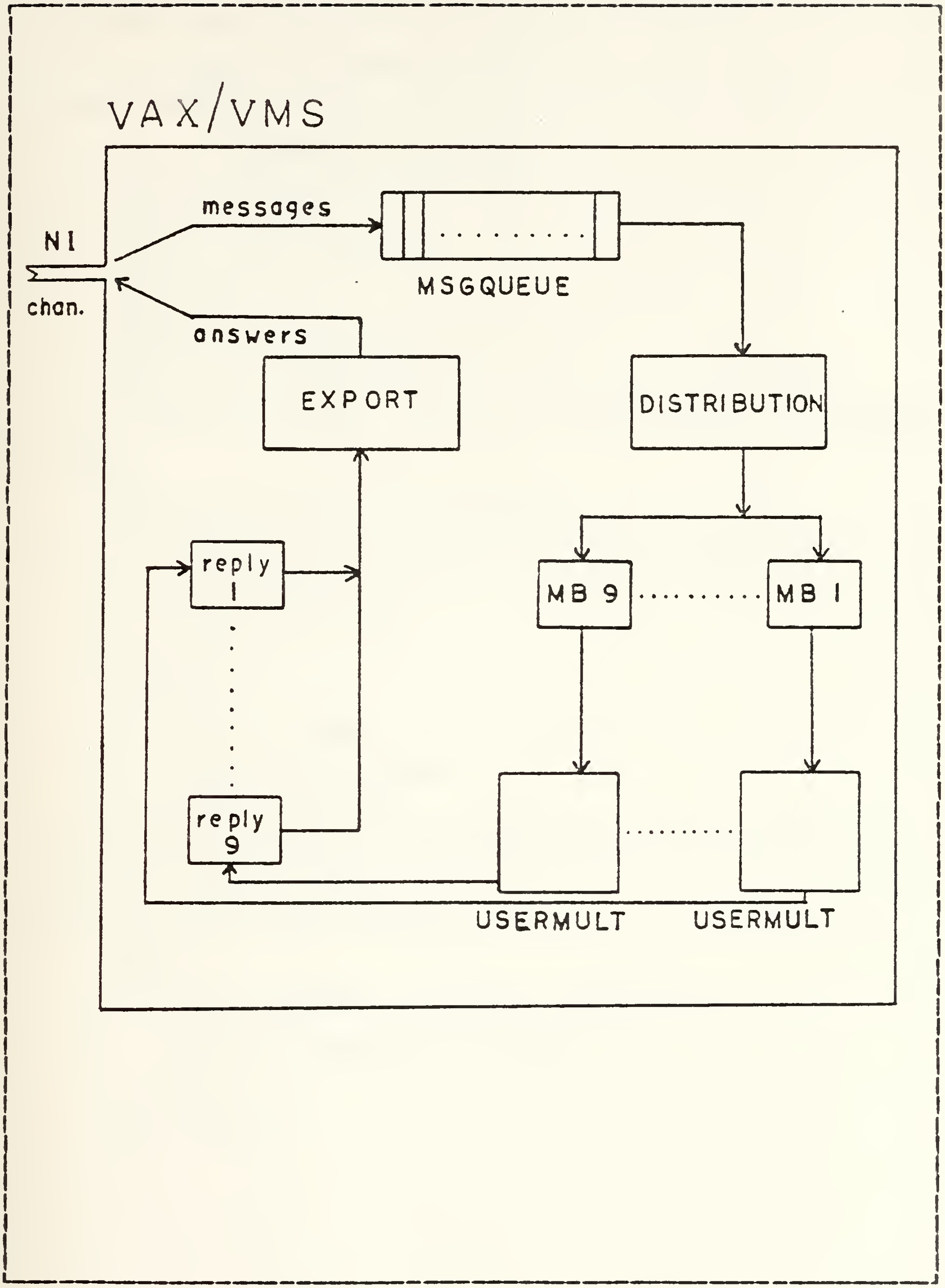

Figure 4.3 Message Traffic Inside the VAX.

in, by making indefinitely invalid selections just before the 10 seconds expire.

Next a concise algorithm for the "Usermult" program follows.

"<u>Usermult</u>" <u>Algorithm</u>

1. Access event flag cluster
2. Check flag #84
3. If set keep checking
   Else set it
4. Check flags #73 through #81
5. Put those flags which are not set in an array
6. Associate these flags with user numbers and prompt the user
7. Set the timer for 10 seconds
8. In case of wrong selection give another chance
9. If time expires without selection of user number
       reset flag #84
       stop
   Else
       cancel timer
       accept user number
       reset flag #84
10. Create mailbox and establish path of communication
11. Check the mailbox
12. If empty keep checking
    Else
        call "Spawn" to execute the message
        put the results in a file
        go to 11
    End Usermult.

V. DETAILED DESIGN OF THE MULTIPLEXING SYSTEM

In the previous chapter the high level design of the multiplexing system was given, without details on the operations of each module.

This chapter explains in detail the operations of the main program and each subroutine. In reality it constitutes the documentation of the software. Also included are comments which explain to the reader why certain decision were made.

A. PROGRAM 'ETHERMULT' DETAILED DESIGN

1. Variables and Data Structures

The variables which are declared in the program are in alphabetical order the following:

Ackflag : Variable to denote whether the received information is an acknowledge or a real message.

Alpha : Array of characters with 9 elements, which contains permanently the numbers from 1 to 9. It is used to solve a technical problem in concatenation of characters. The program assigns very frequently the current user number as an index for the mailboxes and reply files.But in Fortran we are not able to concatenate characters with integers. So the trick which was used was the following: Instead of saying Mailbox//Usernumber which for example may be Mailbox3, we say Mailbox//Alpha(usernumber) that means Mailbox//Alpha(3), which is Mailbox3. The

latter form is acceptable by the Fortran compiler.

C : Number of users currently in the system. It can take values from 1 to 9.

Com : Array of 81 bytes used to store the first 81 symbols of a received Ethernet packet. Its size is adequate to hold any incoming VAX/VMS command.

Condition: Byte used as a boolean variable to denote the existance of a condition.

Dflag : Boolean variable.

Fl : The name of the common block which contains variable ackflag.

Flag : Used as Boolean variable.

Ind1 : Name of the common block in which variable 'Notyet' is found.

Ind2 : Name of the common block which contains variable 'Times'.

Iosb : An output parameter of Sys$qiow routine. Its value determines whether the operation was succesful or not.

Mqueue : Array 9X81 which stores the incoming commands in a FIFO fashion.

MRpacket : Stands for main receive packet. It is an array of 1522 bytes which includes the incoming message plus the header.

N : Variable which denotes the first empty slot of the message queue.

Nichan : Indicates the number of the channel of communication which is assigned to a device by Sys$assign system routine.

Notyet : Array of integers with dimension 1x9. Its elements correspond to one of the nine users and they are used to indicate whether the transmission of all the frames of an answer file of a user has been completed or not.

Pack : Name of common block containing variable 'Mqueue'.

Row : Byte showing the row of the table in which a user number was found.

Rowflag : Byte used as boolean variable.

Slot : Name of common block which contains variable 'N'.

Table : Array of bytes with dimentions 9x4. User information is stored in this table. The first column contains the user number. The second column becomes '1' when the user has an answer that has not been sent yet, and '0' if there is no file with the answer of this user. Columns 3 and 4 contain the source address of the last message addressed to this user.

Times : Array of integers with dimension 1x9. Each one of the nine elements specifies a unit number in which the answer file of one user will be opened. The value of the elements of array 'Times' should be greater than 11.

Usrnum : Byte which stands for user number. It can take values from 1 to 9.

In the program are also included the routines:

_dra0:[npssys.interlan]nidef.for'

This routine contains the definitions of the functions used exclusively by the NI1010 board.

$iodef

This is a macro routine which contains all the definitions of I/O functions [Ref. 6].

$ssdef

Macro routine which contains all the definitions of system status functions.

2. Initializations

The program starts by initializing the number of current users, the user information table,the message queue, and arrays 'Notyet' and 'Times'.

Next the third event flag cluster named NET, which contains the flags #64 through #95 is made visible to the program. Then the flags #64 through #84 are reset to indicate that the system is empty.

Next step is to establish a channel of communication with NI1010 and start up the system. More information about the system routines that execute these tasks can be found in Appendix A.

3. Main Body

At this point the program is ready to start the actual work. The routine REC, which "sets the ear" of the system to the NI1010 board is called. It uses the "Sys$qio" system routine which enables the program to listen to NI1010 without waiting there. While listening, it can proceed with the execution of other operations. As soon as a

message arrives at the NI1010 board, control of the program is transfered to routine "Dummy".

Routine "Dummy" checks whether the received message is a real message or an acknowledge. If the 18th byte of the received packet is "FF" hex, then the messge is an acknowledge. In this case, it "sets the ear" of the system again by calling routine REC, sets the "Ackflag" and returns control at the point where the program was before the interrupt occured.

If the received message was a real one, routine "Xmit" is called to send acknowledge to the source , and the first 81 characters of the packet are transferred into the array "Com". The size of the array "Com" is adequate to hold any one of VAX/VMS commands.

Then the message queue is rearranged and the routine "Formqueue" is called, to put the message in the appropriate place of the queue.

Finally the "ear of the system" is set and control of the program returns where it was before the interrupt occurred.

As the trace of the "Ethermult" continues, the next thing is to determine the user number by examining the 19th byte of the message whose turn is to be processed.

Since user number is of type byte (ASCII) the number 48 is subtracted in order to make it decimal.

After that the obtained user number is checked for validity. If the user number is 0, it is immediately characterized invalid. Subroutine message is called then , to notify the user on the MDS terminal about that. Next, the position of the queue which contains this message is cleared, and the queue is rearranged.

The same things happen when the obtained user number is greater than 9, or of other data type (i.e. character). There is only a small difference in the warning message to the MDS.

If none of the above cases occurs, the number is whithin valid limits and it should be checked whether it already exists in the user information table or not i.e. whether the user is new or he has been already using the system. Subroutine "Search" finds that out. If the number already exists in the user information table, the row in which it is found,is returned.

If the second column of the row that contains the user number is 1, this means that the user hasn't sent yet the entire file with the previous answer. So, no other command should be executed, because the new file which will be created will erase the previous one, before it is sent to the NI1010. Because of that, this user is not served in this round, but the next message in the queue is selected.

If no other ready answer exists, the variable "Condition" becomes 1 in order to terminate the do while loop. Since one message is processed at a time in each round of the program, and a message to be processed has been found, there is no need to continue the while loop.

Next step is to put the message in the "Com" array, and arrange the message queue properly.

Now the message should be placed to the appropriate mailbox. Thus, subroutine "Distribution" is called.

Distribution calls routine "Search" to find out whether the user number exists in the user table. If routine "Search" returns variable "Rowflag" with value 1, it indicates that the user already exists in the table. If also the second column of the row in which the user number was found has value 1 , it means there is still at least a part of the previous file to be sent, so no distribution occurs and control returns to the main program.

If the user number was not found in the user information table, there are two possibilities. Either he is in the system but the table is not updated, or he is not in the system at all.

The answer is given by examining the corresponding common event flag. If the flag #(72+user number) is reset, it means that this user exists in the system and so the message is ignored. If that flag is raised, it means this is a legal user. Then the user information table is updated, a mailbox with the proper index is created, and the message is written in it. The last two operations occur also when the user exists in the system and he has no previous answers to send. Then control returns to main program.

After distribution, subroutine "Export" is called. Routine Export examines the users one by one and determines whether a user has a ready answer to send. This is achieved by reading common event flag #(63+user number). If there is an answer indeed, there are two cases. Either no part of the file has been sent, or some frames have been already transmitted. In the second case, the file should not be reopened, because the read index will be reset at the beginning of the file again, and the track of the transmitted frames will be lost. That's why both conditions are considered in the export routine.

Then the destination address is specified and routine "Sendmsg" is called to send the file.

Sendmsg has as input parameters the file to be sent, the destination address, the user number, the user information table and the number of current users.

First it forms the header of the frame to be transmitted and then clears (ie. initializes to 0) the rest of it. Then it reads the file, which has already been opened in "Export", in frames of 1500 bytes, each time it is called, and transmits them.

Next, it waits for acknowledge from the destination of the answer.

If the last frame of the file was sent (variable Endfil=-1) then the file is closed and the element of array "Notyet" which corresponds to the user number becomes 0.

If this was not the last frame, the file stays opened so that the read index is not removed from the correct position. Then control returns to routine Export and from there back to the main program.

As a final step, the main program calls the subroutine "Status_check" which examines the common flags and if it finds any discrepancy with the existing status, it updates the user information table and the number of current users. Then the program repeats the same cycle idefinitely.

A hierarchy diagram of the program "Ethermult" is given in Figure 5.1.

B. "USERMULT" DETAILED DESIGN

1. Variables and Data Structures

The variables and the data structures which are used in program "Usermult" are given below:

Alpha : Array of dimension 1x9. Its elements are permanently the numbers from 1 to 9. It is used for the same reason as array Alpha in program Ethermult.

Channel : Array 1x9 which contains channel numbers corresponding to user numbers.

Com : Array of 81 bytes used to store a received command (message) from the corresponding mailbox stripped off from its header.

Load : Logic variable. When it is 1 it means the system is full of users.

usernum : User number. Byte which can take values from 1 to 9.

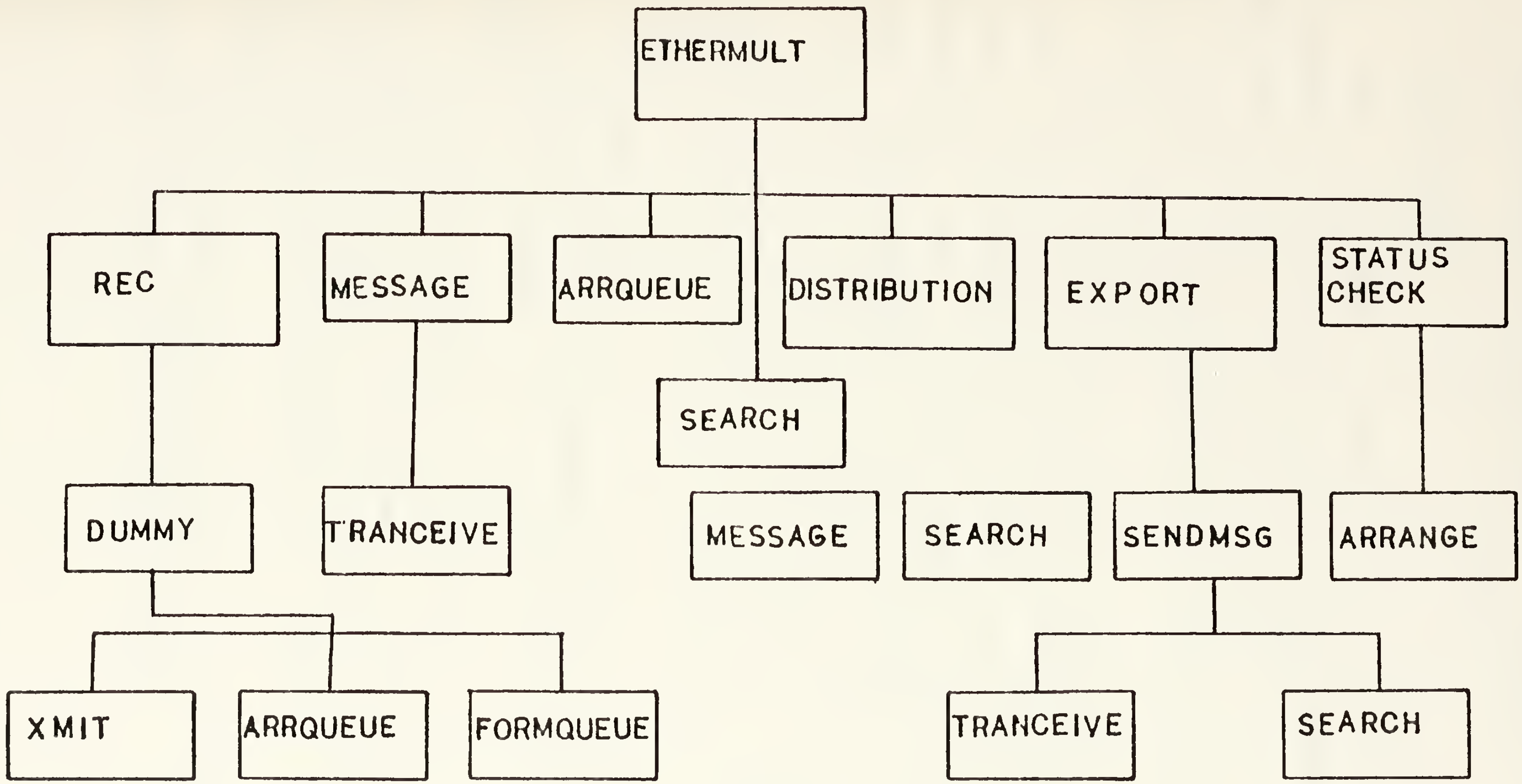

Figure 5.2 Hierarchy Diagram Of Program ETHERMULT

The included macro routine "$iodef"contains all the definitions of I/O functions (see Appendix C).

2. Main Body

The program starts by calling subroutine "Authorize". "Authorize" has as input parameter the user number and returns the variable "Load" with value 1 if the system has already 9 users and it cannot accept any new ones at the present. Detailed description of "authorize" is given in the next subsection.

If the system has accepted the user, the program proceeds with the creation of a user mailbox and establishes a channel of communication with it, using the system routine "sys$crembx". Then it keeps checking the mailbox for any messages. When a message arrives, it reads it and puts it in the array "Com". Routine "Sys$qiow" is used for this task.

Next, the program makes visible to its environment the cluster which starts with flag #64. This is needed because the program must set a certain flag when an answer is ready.

After that, the program calls routine "feedfile". This routine places the command without its header in the file "mail.com" which will be used as input in "Lib$spawn" routine.

Subroutine "Spawn" is called next. "Spawn" has as an input parameter the user number. It executes the command and puts the results in a file which is indexed by the user number. Description of "Spawn" is found in subsection 4 of this section.

After completing this cycle, the program goes back to read the mailbox and proceed as before.

3. Subroutine "Authorize"

This routine checks the common event flags #73 through #81 to find out what user numbers are available for a new user. Then it interacts with the user in order to assign a number to him. If the system is full, it returns the variable "Load" with value 1. In such a case the program terminates execution.

The routine starts by assigning a channel to the terminal. This is done because the "Sys$qiow" routine is used to read a user number from the terminal. Of course, the read statement instead of"Sys$qiow" routine could be used, but in this case the use of the watchdog timer would be impossible. With the read statement, the program would wait indefinitely until a user number was entered. If the user did not enter a number, he could inhibit any other user from logging on. Now however, we are able to set a time limit and if the user does not enter a number within the available time period, the program stops execution.

Next step is to invoke "Sys$qiow" routine to start up the process of I/O.

Then the proper initializations are made. "Flagarray" is an array which will hold the available user numbers in order to display them later on the screen.

Variables "Cancel", "Condition", and "Load" are used as Boolean variables.

Next, the routine aquires acces to the usual common event flag cluster and determines whether another user is using the system (ie. is requesting a user number), by checking flag #84 . If this flag is set, it means the system is busy and the program keeps checking until the flag is reset. The variable "Condition" is used to inhibit the program from writing the message on the screen in each loop.

If the flag #84 is reset that means the system is available. So, immediately the program sets this flag in order to notify the other potential users.

Next step is to check flag #73 through #81. Those flags represent users from 1 to 9. The numbers which are represented by the flags that are reset, are put in the "Flagarray". If after the examination of the flags the "Flagarray" is still empty, this means that the system is full. Then variable "Load" takes value 1 and the routine returns control to main program, after it has printed on the screen the message "system full".

If there are available user numbers, a message is printed on the screen prompting the user to select one of the user numbers which follow on the screen. Immediately the timer is set at 10 seconds, and the "Flagarray" is displayed on the screen.

After that, the program goes to a read state. If a number is entered within 10 seconds the timer is cancelled. If not, control of the program is transferred to routine "Abort" which resets flag #84 and stops execution of the entire program.

After a number is entered, the program checks whether it can be found in the "Flagarray". If it is not found there, the counter advances to count one misselection. The user is notified for his mistake and the program presents again the available user numbers on the screen. Then it proceeds as before. If a second misselection occurs, the program transfers control to subroutine "Abort".

If the entered number is found in the "Flagarray", it is considered a legal one. Then the flag which represents this user number is set, the flag #84 is reset leaving the system available for use by other users, a message is displayed on the screen to notify the user that the number has been accepted and control returns to the main program.

4. <u>Subroutine</u> "<u>Spawn</u>"

Routine "Spawn " executes the DCL command which is entered from an MDS terminal.

It starts by translating the logical name of the input file "Mail.com", into a physical name. This is necessary in order that this file be accepted as input to the "Lib$spawn" procedure.

Then the file is opened and read. Two special commands are examined separately. The "Edit" and "Logout" command. It was found that the "Edit" command in order to function properly must be put explicitly as input parameter in "Lib$spawn". So, the first letters of the file "Mail.com" are examined and if they form the word "Edit", the "Lib$spawn" is called with an input parameter the string "Edit" explicitly and not the file "Mail.com" which contains this string.

The case of "Logout" is examined in order that the termination of the corresponding "Usermult" program occur, and an update of flags is effected. This will enable the main program to update the user information table and the number of current users in the system.

The rest of the commands are executed normally and the results are placed in a file called "Reply" concatenated with a user number. After that control is transferred to the main program.

Figure 5.2 illustrates the hierarchy of "Usermult" program.

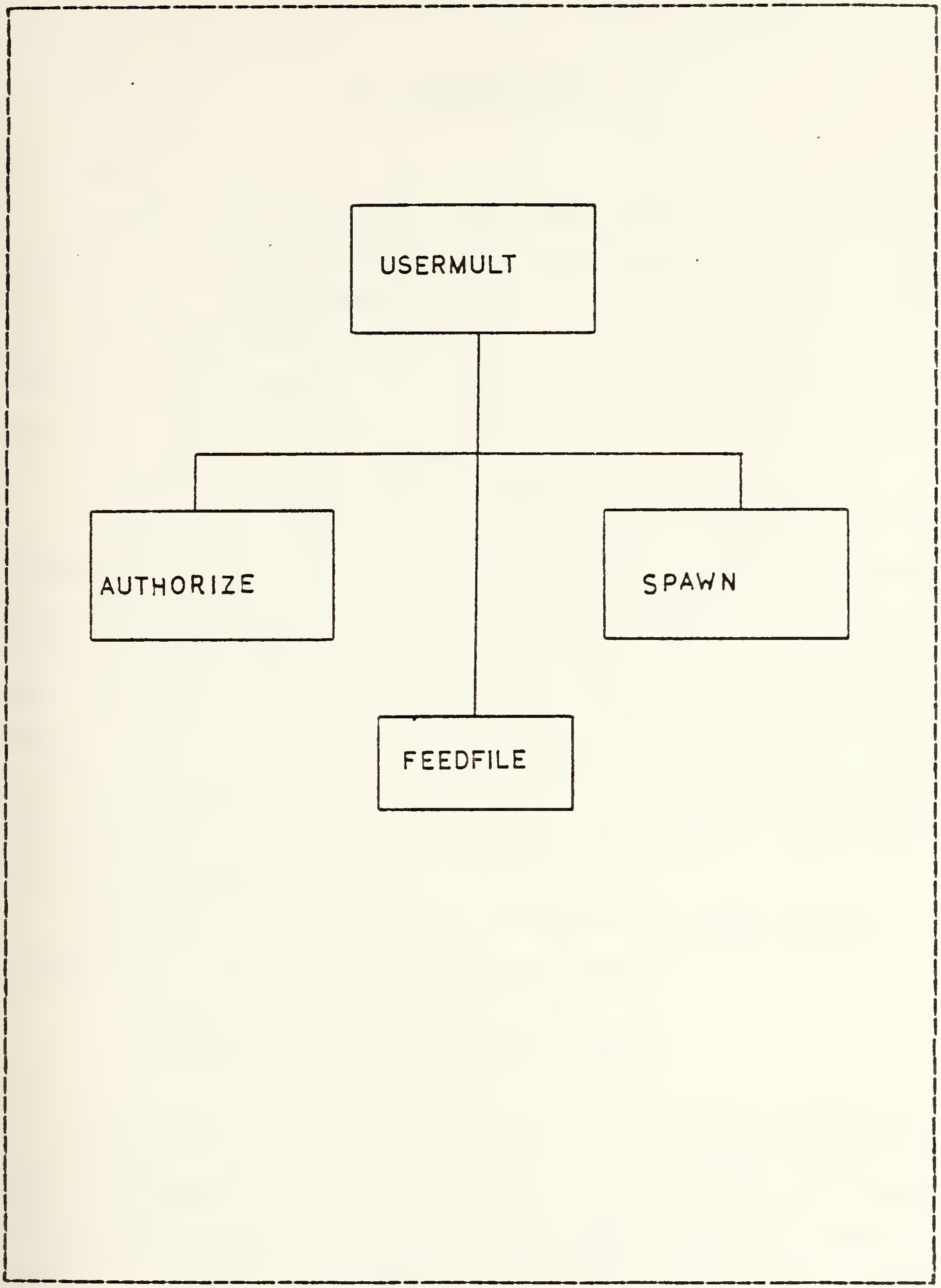

Figure 5.2   Hierarchy Diagram of Program "Usermult".

VI. CONCLUSIONS

A. PRESENT DESIGN

The principal goals of this thesis were met. The multiplexing of the NI1010 board is considered to be efficient and though the developed software can be currently tested with only two users (working on the two MDS terminals), it should perform equally well with all nine users. Of course, more than nine users can be accomodated, if minor changes are introduced in the programs.

At present, the programs "ETHERMULT" and "USERMULT" are available in VAX/VMS public user account under user name "INTERLAN" with password "VMS". The NI1010 Ethernet Controller Multiplexing User's Manual is also available in the file NIMUX.DAT. The content of this file is exactly the same as the content in Appendix D of this thesis. Users who want to have a feeling of how this multiplexing system works, can get a hard copy of this file by simply logging into the VAX/VMS under user name and password mentioned above and printing the files. Then the steps in the manual must be followed.

The files containing the software of this thesis and residing in the public user account are:

ETHERMULT.FOR (source code)
ETHERMULT.EXE (executable code)

This is the main program which does the multiplexing of the NI1010 board. When it is run, it puts the NI1010 board in a receive mode and starts a loop until a message (currently originating from an MDS terminal), is received. If this message is properly encoded and contains a legiti-

mate frame, the system responds by transmitting via the NI1010 an answer to the originator. If this is a bad frame it discards it.

The answer transmitted may be:

(a). One message (frame) to the originator stating either that he has not the authority to enter the system or that his message has not the proper form.

(b). A series of one or more frames containing the answer to his message.

USERMULT.FOR (source code)

USERMULT.EXE (executable code)

This is the program which must be running for each user in the system. It takes care of the user's requests, by producing an answer file, which then is transmitted by the "ETHERMULT" via Ethernet to the originator of the request. It also contains the code that provides authorization for each new user. It can be terminated by the remote user when the command "LOGOUT" or "LO" is sent.

B. FUTURE DESIGN

The above programs, provide the possibility for several remote users to take advantage of the majority of the VAX/VMS facilities. These MDS terminals, executing VMS commands, act like virtual terminals of VAX/VMS. To achieve a full virtual terminal performance though, a user should be able to:

1. Use the $CREPRC system service to create a <u>detached</u> process that will execute the LOGINOUT procedure. As "input" and "output" parameters of this process should be defined, instead of a physical VAX terminal, two files, one for the input and the other for the output of the LOGINOUT. Thus,

it will be possible for the LOGINOUT procedure to be invoked from I/O devices other than a VAX terminal (e.g. MDS terminal). This should be feasible with the new version of VAX/VMS. The program which has been worked out by the authors of this thesis during the time period they were trying to achieve this goal, is presented in Appendix I.

2. Execute the LOGINOUT.EXE file, for logging in the VAX/VMS system, from his terminal without having to run the "Usermult" program from a VMS terminal.

3. Change the "Usermult" program by deleting the "SPAWN" subroutine since, if he succeeds in the above two goals he will have direct access to the command language interpreter. Thus, every VMS command would be executed as if it was typed from a real VAX/VMS terminal. The only difference would be that the input and output of the LOGINOUT procedure would not be a terminal but a file.

4. Special privileges must be granted to "Ethermult" program in order to have access to the various directories where "REPL.DAT" files are generated. In cooperation with VAX-11 professional staff, it can be determined which privileges are necessary.

5. Introduce a variable length frame for the network communications, for the sake of efficiency. This would imply changes to both this and Stotzer's thesis' programs. For this thesis the change visualized is:

Instead of declaring a frame size of 1500 bytes as appears in the present form of the programs, a common variable denoting the frame size would have to be introduced. The subroutine that reads a file into packets ("Sendmsg") should be partly revised so that it "tailors" a frame's size according to the available data in the file (46 to 1500 Bytes per frame). For example, in a 2000-bytes file the

first frame should be 1500 bytes since there are 1500 bytes of data available at that time. The second frame though, should be filled-up with 500 bytes only since that much data is left. The "Endfile" condition in Fortran will be helpful on that since, when the "end of file" is reached, the control can be trasferred to another point of the program. At that point, a count of 500 bytes should be performed and the value of that count, namely 500, would be passed to the common variable of the frame size. In such a scheme, a relative flexibility is achieved and the response time of the network is improved.

These are the most important changes that, in the authors' opinion, are needed to support a virtual terminal design.

APPENDIX A

SYSTEM SERVICES AND RUN-TIME LIBRARY ROUTINES

A. CALLING THE SYSTEM SERVICES

This section provides an overview of the calling mechanism for the system services and library routines. This mechanism varies from one language to another. Since the FORTRAN is the language used for this thesis, we will deal only with the FORTRAN's calling mechanism.

1. Fortran Calls

(a). All subprogram calls, including system services and library routines use a CALL instruction.

(b). System services can be called as functions or as subroutines.

(c). Subroutines do not return a status value, therefore they are rarely appropriate.

Example:
```
CALL SYS$serv_name(arg1,arg2,...,argn)
```

(d). Functions return a status value as the function result. The function and the variable to contain the return function value must be declared as INTEGER*4.

Example:
```
INTEGER*4   SYS$serv_name, ret_val
                .
                .
                .
ret_val=SYS$serv_name(arg1,arg2,...,argn)
```

(e). All arguments are positional and must be present. Even if the arguments are optional, their position must be denoted as empty.

Example:
```
ret_val = SYS$ASSIGN('NIAO',NICHAN,,)
```

Two sample FORTRAN calls follow :

-- Calling services as subroutines :

```
IEFN = 1
CALL SYS$CLREF(%VAL(IEFN))
```

-- Calling services as functions :

```
INTEGER*4    SYS$CLREF,STAT
                .
                .
                .
IEFN = 1
STAT = SYS$CLREF(%VAL(IEFN))
```

2. <u>Passing Arguments</u>

There are three ways to pass arguments to system services:

(a). <u>By immediate value</u>. The argument is the actual value to be passed (a number or a symbolic representation of a numeric value).

-- Example:

```
IEFN = 1
    .
    .
    .
CALL SYS$CLREF(%VAL(IEFN))
```

(b). <u>By reference</u>. The address of the argument is passed in the argument list.

-- Example :

```
INTEGER*4   NICHAN
      .
      .
      .
    ISTAT = SYS$ASSIGN('NIA0',NICHAN,,)
or: ISTAT = SYS$ASSIGN('NIA0',%REF(NICHAN),,)
```

(c). <u>By descriptor</u>. The address of a structure, called <u>a character string descriptor</u>, describing the data, is placed in the argument list.

```
-- Example : INTEGER*2     W_NAMELEN
             INTEGER*4     SYS$TRNLOG, TRN_STATUS
             CHARACTER_63 EQV_BUFFER
                      .
                      .
                      .
             TRN_STATUS    =     SYS$TRNLOG('LOGNAM',
W_NAMELEN,EQV_BUFFER,,,)
```

Here, "LCGNAM" is passed by descriptor.

3. <u>Testing Return Status Codes</u>

If a function call to a system service is used, then it is possible to check if this service was executed correctly or not. This happens because the return status variable contains now a special code which can be examined. This code is normally contained in the VAX/11 general purpose registers R0/R1 in the form shown in Figure A.1.

31 – 27	27 – 15	15 – 02	02 – 00
CONTROL	FACILITY NUMBER	MSG NUMBER	SEVERITY

Figure A.1 Layout of Status Value (R0 Register).

The severity codes, determined from the three least significant bits of R0 are :

0	Warning
1	Success
2	Error
3	Information
4	Severe error

5-7 (not used).

We can test for successful completion if we test the R0<0>, ie the LSB of the longword. If it is set then the service was completed successfuly. We accomplish this check by treating the return status as logical variable (although declared as INTEGER*4). It is TRUE if the LSB is 1, FALSE if it is 0.

<u>Example</u>:

```
INTEGER*4   STAT, SYS$ASSIGN
     .
     .
     .
STAT = SYS$ASSIGN('NIAO',NICHAN,,)
IF (.NOT. STAT) handle the error
```

To test for a specified error, test R0<31:0> ie the whole longword. All system facilities (corresponding to each kind of error) have their own symbolic error codes. Their names can be found in VAX/VMS System Service Reference Manual, Appendix A, Section A.7. [Ref. 6]. Also, these codes are defined in the $SSDEF macro.

<u>Example</u>:

```
IMPLICIT   INTEGER*4(a-z)
INCLUDE    '($SSDEF)'
     .
     .
     .
STATUS = SYS$SETEF(%VAL(1))
IF (STATUS.EQ.SS$_WASSET)    e.t.c.
```

B. SYSTEM SERVICES AND LIBRARY ROUTINES USED IN THE PROGRAMS

This section is not intended to examine each of the above services/routines in great detail since, much information can be found in the VAX/VMS manuals. Instead, a brief description of the main features is given, along with particular points of attention for their efficient use.

In the VAX/VMS high level languages the system services appear in the form: SYS$service_name; eg, the service $ASSIGN is written as SYS$ASSIGN. Also, the Run-Time Library Procedures have the form LIB$procedure_name; eg, the procedure $SPAWN is written as LIB$SPAWN.

Almost all of the above services/procedures are accompanied by arguments, either optional or mandatory, which represent the neccessary information they need to carry-out the required task. Optional arguments are denoted with their names in angle brackets, mandatory ones are denoted without brackets.

1. System Service Routines

$ASCEFC -- Associate Common Event Flag Cluster

The Associate Common Event Flag Cluster system service causes a named common event flag cluster to be associated with a process for the execution of the current image and assigned a process-local cluster number for use with other event flag services. If the named cluster does not exist but the process has suitable privilege, the service creates the cluster.

High Level Language Format:

<u>SYS$ASCEFC</u>(efn,name,<prot>,<perm>)

<u>efn</u>

Number of any event flags in the common cluster to be associated. The flag number must be in the range of 64 through 95 for cluster 2 and 96 through 127 for cluster 3.

<u>name</u>

Address of a character string descriptor pointing to the text name string for the cluster. Section 3.7.1 of the System Services Reference Manual [Ref. 6] explains the format of this string. The names of event flag clusters are unique to UIC groups.

<u>prot</u>

Protection indicator controlling group access to the common event flag cluster. A value of 0 (default) indicates that any process in the creator's group may access the cluster. A value of 1 indicates that access is restricted to processes executing with the creator's UIC.

<u>perm</u>

Permanent indicator. If it is 1, the common event cluster is marked permanent. If it is 0 the cluster is temporary (this is the default value).

<u>Privilege restrictions</u>

To create a permanent common event flag cluster, the user privilege PRMCEB is required. To create a common event flag cluster in memory shared by multiple processors, the user privilege SHMEM is required.

Resources required/returned:

Creation of temporary common event flag clusters uses the process' quota (TQELM); the creation of a permanent cluster does not affect the quota.

Notes

(1). Temporary clusters are automatically deleted when the image that created them, exits.

(2). Since this service automatically creates the common event flag cluster if it does not already exist, cooperating processes need not be concerned with which process executes first to create the cluster. The first process to call $ASCEFC creates the cluster and the others associate with it regardless of the order in which they call the service.

(3). The initial state of all event flags in a newly created common event flag cluster is 0.

(4). If a process has already associated a cluster number with a named common event flag cluster and then issues another call to $ASCEFC with the same cluster number, the service disassociates the number from it's first assignment before associating it with it's second.

$ASSIGN -- Assign I/O channel

The Assign I/O Channel system service provides a process with an I/O channel so that input/output operations can be performed on a device, or establishes a logical link with a remote node on a network.

High level language format:

<u>SYS$ASSIGN</u> (devnam,chan,<acmode>,<mbxnam>)

<u>devnam</u>

Address of a character string descriptor pointing to the device name string. The string may be either a physical device name or a logical name. If the device name contains a colon, the colon and the characters that follow it, are ignored. If the first character in a string is an underscore (_), the name is considered a physical device name. Otherwise the name is considered a logical name and logical name translation is performed until either a physical device name is found or the system default number of translations has been performed.

<u>chan</u>

Address of a word to receive the channel number assigned.

<u>acmode</u>

Access mode to be associated with the channel. The most privileged access mode used is the access mode of the caller. I/O operations on the channel can only be performed from equal and more privileged access modes.

<u>mbxname</u>

Address of a character string descriptor pointing to the logical name string for the mailbox to be associated with the device, if any. The mailbox receives status information from the device driver.

<u>Privilege restrictions</u>

The NETMBX privilege is required to perform network operations.

Notes

(1). Only the owner of a device can associate a mailbox with the device, and only one mailbox can be associated with the device at a time.

(2). Channels remain assigned until they are explicitly deassigned with the Deassign I/O channel ($DASSGN) system service, or, if they are user mode channels, until the image that assigned the channel is terminated.

$BINTIM -- Convert ASCII String to Binary Time

The Convert ASCII String to Binary Time converts an ASCII string to an absolute or delta time value in the system 64-bit time format suitable for input to the Set Timer ($SETIMR) or Schedule Wakeup ($SCHDWK) system services.

High level language format:

SYS$BINTIM (timbuf,timadr)

timbuff

Address of a character string descriptor pointing to the buffer containing the absolute or delta time to be converted. The required formats of the ASCII strings and syntax rules along with several examples, are described in the VAX/VMS System Services Reference Manual [Ref. 6].

Notes

(1). The $BINTIM service executes at the access mode of the caller and does not check whether address arguments

are accessible before it executes. Therefore, an access violation causes an exception condition if the input buffer descriptor cannot be read or the output buffer cannot be written.

(2). This service does not check the length of the argument list, and therefore cannot return the SS$_INSFARG (insufficient arguments) error status code. If the service does not receive enough arguments (for example you omit required commas in the call), you might not get the desired result.

<u>$CANTIM</u> -- Cancel Timer

The Cancel Timer Request system service cancels all or a selected subset of the Set Timer ($SETIMR) requests previously issued by the current image executing in a process. Cancellation is based on the request identification specified in the $SETIMR system service. If more than one timer request was given the same request identification, they are all canceled.

High Level Language format:

<u>SYS$CANTIM</u> (<requidt>,<acmode>)

<u>reqidt</u>

Request identification of the timer request(s) to be canceled. A value of 0 (the default) indicates that all timer requests are to be canceled.

acmode

Access mode of the request(s) to be canceled. The most privileged access mode used is the access mode of the caller. Only those timer requests issued from an access mode equal to or less privileged than the resultant access mode are canceled.

Privilege Restrictions

Timer requests can be canceled only from access mode equal or more privileged than the access mode from which the requests were issued.

Resources Required/Returned

Cancelled timer requests are restored to the process' quota for timer queue entries (TQELM quota). Outstanding timer requests are automatically cancelled at image exit.

$CLREF -- Clear Event Flag

The Clear Event Flag system service sets an event flag in a local or common event flag cluster to 0.

High Level Language Format

SYS$CLREF (efn)

efn

Number of the event flag to be cleared.

$CREMBX -- Create Mailbox and Assign Channel

The Create Mailbox and Assign Channel system service creates a virtual mailbox device named MBAn: and assigns an I/O channel to it. The system provides the unit number, n, when it creates the mailbox. If a mailbox with the specified name exists, the $CREMBX service assigns a channel to the existing mailbox.

High Level Language Format:

SYS$CREMBX (<prmflg>, chan, <maxmsg>, <bufquo>, <promsk>, <acmode>, <lognam>)

prmflg

Permanent indicator. A value of 1 indicates that a permanent mailbox is to be created. The logical name, if specified, is entered in the system logical name table. A value of 0 (the default) indicates a temporary mailbox.

chan

Address of a word to receive the channel number assigned.

maxmsg

Number indicating the maximum number of messages that can be sent to the mailbox. If not specified, or is specified as 0, the system provides a default value.

bufquo

NUmber of bytes of system dynamic memory that can be uset to buffer messages sent to the mailbox. If not specified, or if specified as 0, the system provides a default value.

promsk

Numeric value representing the protection mask for the mailbox. The mask contains four 4-bit fields. Bits are read from right to left in each field. If not specified, or specified as 0, read and write privilege is granted to all users.

acmode

Access mode to be associated with the channel to which the mailbox is assigned. The most privileged access mode is the mode of the caller.

lognam

Address of a character string descriptor pointing to the logical name string for the mailbox. The equivalence name for the mailbox is MBAn:. The first character in the equivalence name string is an underscore character (_). One or more processes use the mailbox to assign other I/O channels to the mailbox.

Privilege Restrictions

The user privileges TMPMBX and PRMMBX are required to create temporary and permanent mailboxes, respectively.

The user privilege SHMEM is required to create a mailbox in memory shared by multiple processors.

Resources Required/Returned

(1). System dynamic memory is required for the allocation of a device data base for the mailbox and for an entry in the logical name table, if a logical name is specified.

(2). When a temporary mailbox is created, the process' buffered I/O byte count quota (BYTLM) is reduced by

the amount specified in the BUFQUO argument. The size of the mailbox unit control block and the logical name (if one is specified), are also subtracted from the quota. The quota is returned to the process when the mailbox is deleted.

Notes

(1). After a mailbox is created, the creating process and other processes can assign additional channels to it by calling the Assign I/O Channel ($ASSIGN) or Create Mailbox ($CREMBX) system services. The system maintains a reference count of a number of channels assigned to a mailbox; the count is decreased whenever a channel is deassigned with the Deassign I/O Channel ($DASSGN) system service or when the image that assigned the channel terminates. If it is a temporary mailbox, it is deleted when there are no more channels assigned to it.

(2). A mailbox is treated as a shareable device; it cannot, however, be mounted or allocated. In other words, it cannot be reserved for exclusive use (allocated) or cannot be linked with a volume and a process (mounted).

(3). $CREMBX merely assigns a channel if the mailbox already exists in order to remove the need for cooperating processes to consider which process must execute first to create the mailbox. If a temporary mailbox is being created, $CREMBX implicitly qualifies the mailbox name with the group number to check whether the mailbox already exists. In other words, there can be only one mailbox per group with the same name. For permanent mailboxes, there can be only one mailbox with a particular name. However, a permanent mailbox and group of mailboxes can have the same name.

$CREPRC -- Create Process

The Create Process system service allows a process to create another process. The created process can be either a subprocess or a detached process.

A detached process is a fully independent process. For example, the process that the system creates when a user logs in is a detached process. A subprocess, on the other hand, is related to its creator in a tree-like structure; it receives a portion of the creating process' quotas and must terminate before the creating process. The specification of the UIC argument controls whether the created process is a subprocess or a detached process.

High Level Language Format:

SYS$CREPRC (<pidadr>, image, <input>, <output>, <error>, <prvadr>, <quota>, <prcnam>, <baspri>, <uic>, <mbxunt>, <stsflg>)

pidadr

Address of a longword to receive the process identification number assigned to the created process.

image

Address of a character string descriptor pointing to the file specification of the image to be activated in the created process. The image name can have a maximum of 63 characters. If the image name contains a logical name, the equivalence name must be in a logical name table that can be accessed by the created process.

input

Address of a character string descriptor pointing to the equivalence name string to be associated with the

logical name SYS$INPUT in the logical name table for the created process. The equivalence name string can have a maximum of 63 characters.

<u>output</u>

Address of a character string descriptor pointing to the equivalence name SYS$OUTPUT in the logical name table for the created process. The equivalence name string can have a maximum of 63 characters.

<u>error</u>

Address of a character string descriptor pointing to the equivalence name string to be associated with the logical name SYS$ERROR in the logical name table for the created process. The equivalence name string can have a maximum cf 63 characters.

<u>prvadr</u>

Address of an 64-bit mask defining privileges for the created process. The mask is formed by setting the bits corresponding to specific privileges. The $PRVDEF macro defines the symbolic names for the bit offsets. For more information see the VAX/VMS System Services Reference Manual [Ref. 6].

<u>quota</u>

Address of a list of values assigning resource quotas to the created process. If no address is specified, or the address is specified as 0, the sustem supplies default values for the resource quotas.

<u>prcnam</u>

Address of a character string descriptor pointing to a 1- to 15-character process name string to be assigned to the created process. The process name is implicitly qualified by the group number of the caller, if a subprocess is

created, or by the group number in the UIC argument, if a detached process is created.

baspri

Numeric value indicating the base priority to be assigned to the created process. The priority must be in the range of 0 to 31, where 31 is the highest priority level and 0 is the lowest. Normal priorities are in the range 0 through 15, and real-time priorities are in the range 16 through 31.

The user privilege SETPRI is required to set a priority higher than one's own. If the caller does not have this privilege, the specified base priority is compared with the caller's priority and the lower of the two values is used.

uic

Numeric value representing the user identification code (UIC) of the created process. This argument also indicates whether a process is a subprocess or a detached process|. If not specified, or specified as 0 (the default), it indicates that the created process is a subprocess; the subprocess has the same UIC as the creator.

The specified value is interpreted as a 32-bit octal number, with two 16-bit fields:

bits 0-15 - member number
bits 16- 31 - group number

The user privilege DETACH is required to create a detached process.

mbxunt

Unit number of a mailbox to receive a termination message when the created process is deleted. If not specified, or specified as 0 (the default), the system sends no

termination message when it deletes the process. The format of the message is described in Note 2 below. The Get Device/Volume Information ($GETDVI) system service must be used to obtain the unit number of the mailbox.

<u>stsflg</u>

32-bit status flag indicating options selected for the created process. The flag bits, when set, have the described in VAX/VMS System Service Manual p.47 [Ref. 6].

<u>Privilege Restrictions</u>

User privileges are required to:

- Create detached processes (DETACH privilege).

- Set a created subprocess's base priority higher than one's own (ALTPRI privilege).

- Grant a process user privileges that the caller does not have (SETPRV privilege).

- Disable either process swap mode (PSWAPM privilege) or accounting functions (NOACNT privilege) for the created process.

- Create a network connect object (NETMBX privilege).

<u>Resources Required/Returned</u>

(1). The number of subprocesses that a process can create is controlled by the sybrocess quota (PRCLM); the quota amount is returned when a subprocess is deleted.

(2). The Create Process system service requires system dynamic memory.

(3). When a subprocess is created, the value of any deductible quota is subtracted from the total value the creator has available; and when the subprocess is deleted, the unused portion of any deductible quota is added back to the total available to the creator. Any pooled quota value is shared by the creator and all its subprocesses.

Notes

(1). Some error conditions are not detected until the created process executes. These conditions include an invalid or nonexistent image; invalid SYS$INPUT, SYS$POUTPUT, or SYS$ERROR logical name equivalence; and inadequate quotas or insufficient privilege to execute the requested image.

(2). If mailbox unit is specified, the mailbox is not used until the created process actually terminates. At the time, an $ASSIGN system service is issued for the mailbox in the context of the terminating process and an accounting message is sent to the mailbox. If the mailbox no longer exists, cannot be assigned, or is full, the error is treated as if no mailbox had been specified.

(3). All subprocesses created by a process must terminate before the creating process can be deleted. If subprocesses exist when their creator is deleted, they are automatically deleted.

(4). A detached process cannot run an image containing a call to the Run-Time Library procedure LIB$DO_COMMAND; this restriction exists because no CLI is defined when the new process is created.

$QIO- QUEUE I/O REQUEST

The Queue I/O Request system service initiates an input or output operation by queueing a request to a channel associated with a specific device. Control returns immedately to the issuing process, which can synchronize I/O completion in one of the three ways:

1. Specify the address of an AST routine that is to execute when the I/O completes.

2. Wait for a specified event flag to be set.

3. Poll the specified I/O status block for a completion status.

When the service is invoked, the event flag is cleared (event flag 0, if not specified); if the IOSB argument is specified, the I/O status block is cleared.

High-level Language Format

SYS$QIO(<efn> ,chan ,func ,<iosb> ,<astadr> ,<astprm>
 ,<p1> ,<p2> ,<p3> ,<p4> ,<p5> ,<p6>)

efn

Number of the event flag that is to be set at requested completion. if not specified, it defaults to 0.

chan

Number of the I/O channel assigned to the device to which the request is directed.

func

Function code and modifier bits that specify the operation to be performed. The code is expressed symbolically. For reference purposes, the function codes are listed

in VAX/VMS System Service Manual Appendix A, Section A.2. Complete details on valid I/O function codes and parameters required by each are documented in the <u>VAX/VMS I/O User's Guide</u>.

<u>iosb</u>

Address of a quadword I/O status block that is to receive final completion status.

<u>astadr</u>

Address of the entry mask of an AST service routine to be executed when the I/O completes. If specified, the AST routine executes at the access mode from which the $QIO service was requested.

<u>astprm</u>

AST parameter to be passed to the AST service routine.

<u>p1 to p6</u>

Optional device- and function-specific I/O request parameters.

<u>Privilege Restrictions</u>

The Queue I/O Request system service can be performed only on assigned I/O channels and only from access modes that are equal to or more privileged than the access mode from which the original channel assignement was made.

<u>Resources Required/Returned</u>

(1). Queued I/O requests use the process's quota for buffered I/O (BIOLM) or direct I/O (DIOLM); the process's

buffered I/O byte count (BYTLM) quota; and, if an AST routine is specified, the process's AST limit quota (ASTLM).

(2). System dynamic memory is required to construct a data base to queue the I/O request. Additional memory may be required on a device-dependent basis.

Notes

(1). The specified event flag is set if the service terminates without queuing an I/O request.

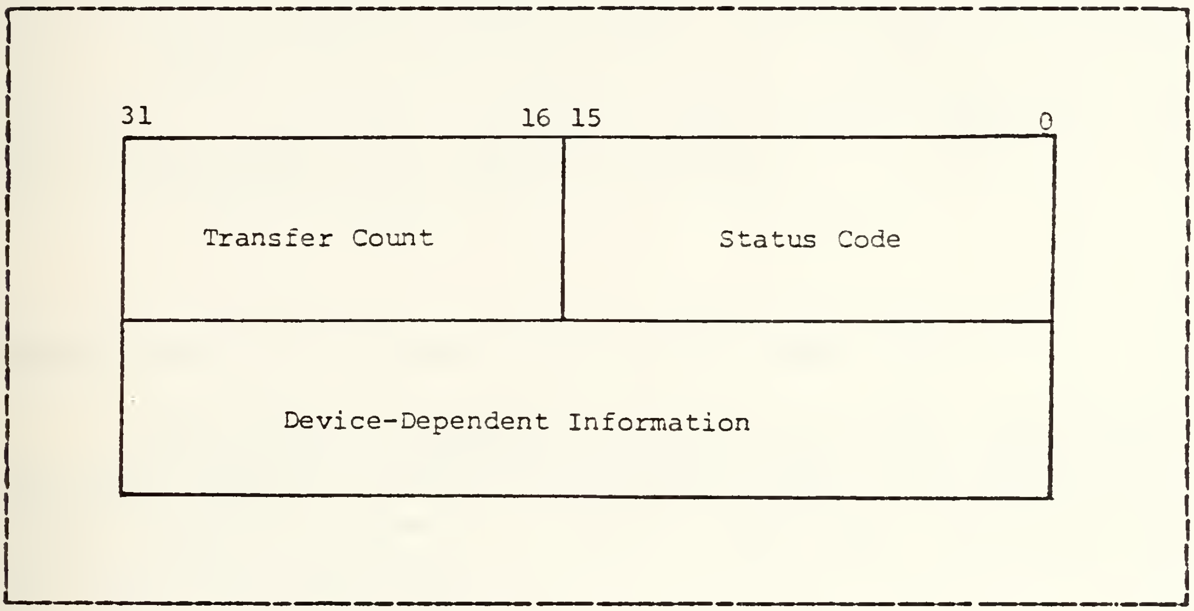

Figure A.2 I/O Status Block.

(2). The I/O status block has the following format:

status

Completion status of the I/O request.

byte count

Number of bytes actually transerred. Note that for some devices this contains only the low-order word of the count. For information on specific devices, see the *VAX/VMS I/O User's Guide*.

Device-and function-dependent information varies according to the device and operation being performed. The information returned for each device and function code is documented in the *VAX/VMS System Services, I/O User's Guide* [Ref. 11].

(3). Many services return character string data and write the length of the data returned in a word provided by the caller. Function codes for the $QIO system service (and the LENGTH argument of the $OUTPUT system service) require length specifications in longwords. If lengths returned by other services are to be used as input parameters for $QIO requests, a longword should be reserved to ensure that no error occurs when $QIO reads the length.

$QIOW- Queue I/O Request and Wait for Event Flag

The Queue I/O Request and Wait for Event Flag system service combines the $QIO and $WAITFR (Wait for Single Event Flag) system services. It can be used when a program must wait for I/O completion.

High-level Language Format

SYS$QIOW(<efn> ,chan ,func ,<iosb> ,<astadr> ,<astprm>, <p1> ,<p2> ,<p3> ,<p4> ,<P5> ,<p6>)

efn

Number of the event flag that is to be set at request completion. If not specified, it defaults to 0.

<u>chan</u>

Number of the I/O channel assigned to the device to which the request is directed.

<u>func</u>

Function code and modifier bits that specify the operation to be performed. The code is expressed symbolically.

<u>iosb</u>

Address of a quadword I/O status block that is to receive final completion status.

<u>astadr</u>

Address of the entry mask of an AST service routine to be executed when the I/O completes. If specified, the AST routine executes at the access mode from qhich the $QIO service was requested.

<u>astprm</u>

AST parameter to be passed to the AST completion routine.

p1 to p6

Optional device-and function-specific I/O request parameters.

The first parameter may be specified as P1 or P1V, depending on whether the function code requires an address or a value, respectively. If the keyword is not used, P1 is the default; that is, the argument is considered an address.

P2 through Pn are always interpreted as values.

Privilege Restrictions

See the description of the $QIO system service for details.

Resources Required/Returned

See the description of the $QIO system service for details.

Notes

See the description of the $QIO system service for details.

$READEF - Read Event Flags

The Read Event Flags system service returns the current status of all 32 flags in a local or common event flag cluster.

High-level Language Format

SYS$READEF(efn ,state)

efn

Number of any event flag within the cluster to be read. A flag number of 0 through 31 specified cluster 0, 32 through 63 specifies cluster 1. and so forth.

state

Address of a longword to receibe the current status of all event falgs in the cluster.

$SETEF - Set Event Flag

The Set Event Flag system service sets an event flag in a local or common event flag cluster to 1. Any processes waiting for the event flag resume execution.

High-level Language Format

SYS$SETEF(efn)

efn

Number of the event flag to be set.

$SETIMR- Set Timer

The Set Timer system service allows a process to schedule the setting of an event flag and/or the queuing of an AST at some future time. The time for the event can be specified as an absolute time or as a delta time.

When the service is invoked, the event flag is cleared (event flag 0 is cleared, if none is specified).

High-level Language Format

SYS$SETIMR(<efn> ,daytim ,<astadr> ,<requidt>)

efn

Event flag number of the event flag to set when the time interval expires. If not specified, it defaults to 0.

daytim

Address of the quadword expiration time. A positive time value indicates an absolute time at which the timer is to expire. A negative time value indicates an offset (delta time) from the current time.

astadr

Address of the entry mask of an AST service routine to be called when the time interval expires. If not specified, it defaults to 0 ,indicating no AST is to be queued.

reqidt

Number indicating a request identification. If not specified, it defaults to 0. A unique request identification can be specified in each set timer request, or the same identification can be given to related timer requests. The identification can be used later to cancel the timer request(s). If an AST service routine is specified, the identification is passed as the AST parameter.

Resources Required/Returned

(1). The Set Timer system service requires dynamic memory.

(2). The Set Timer system service uses the process's quota for timer queue entries (TQELM) and, if an AST service routine is specified, the process's AST limit quota (ASTLM).

notes

(1). The access mode of the caller is the access mode of the request and of the AST.

(2). If a specified absolute time value has already passed, the timer expires at the next system clock cycle (that is, within 10 milliseconds).

(3). The Convert ASCII String to Binary Time ($BINTIM) system service converts a specified ASCII string to the quadword time format required as input to the $SETIMR service.

$TRNLOG-Translate Logical Name

The Translate Logical Name system service searches the logical name tables for a specified logical name and returns an equivalence name string. The process, group ,and system logical name tables are searched in that order.

The first string match returns the equivalence string into a user-specified buffer; the search is not iterative.

High-level Language Format

SYS$TRNLOG(lognam,<rsllen>,
rslbuf,<table>,<acmode>,<dsbmsk>)

lognam

Address of a character string descriptor pointing to the logical name string.

rsllen

Address of a word to receive the length of the translated equivalence name string

rslbuf

Address of a character string descriptor pointing to the buffer that is to receive the resultant equivalence name string.

table

Address of a byte to receive the number of the logical name table in which the match was found. A return value of 0 indicates that the logical name was found in the system logical name table; 1 indicates the group table, and 2 indicates the process table.

acmode

Address of a byte to receive the access mode from which the logical name table entry was made. Data received in this byte is valid only if the logical name match was found in the process logical name table.

dsbmsk

Mask in which bits set to 1 disable the search of particular logical name tables. If bit 0 is set, the system logical name table is not searched; if bit 1 is set ,the group logical name table is not searched; if bit 2 is set, the process logical name table is not searched.

If no mask is specified or is specified as 0 (the default), all three logical name tables are searched.

Notes

If the first character of a specified logical name is an underscore character (_), no translation is performed. However, the underscore character is removed from the string and the modified string is returned in the output buffer.

$WAITFR- Wait for Single Event Flag

The Wait for Single Event Flag system service tests a specific event flag and returns immediately if the flag is set. Otherwise, the process is placed in a wait state until the event flag is set.

High-level Language Format

SYS$WAITFR(efn)

efn

Number of the event flag for which to wait.

Notes

The wait state caused by this service can be interrupted by an asynchronous system trap (AST) if (1) the access mode at which the AST executes is more privileged than or equal in privilege to the access from which the wait was issued and (2) the process is enabled for ASTs at that access mode.

When the AST service routine completes execution, the system repeats the $WAITFR request. If the event flag has been set, the process resumes execution.

LIB$SPAWN- Spawn a Subprocess

LIB$SPAWN requests the calling process's Command Language Interpreter (CLI) to spawn a subprocess for executing CLI commands. LIB$SPAWN provides the same function as the DCL SPAWN command. The subprocess inherits the following from the caller's environment:

- Process logical names

- Global and local CLI symbols

- Default device and directory

- Process privileges

- Process nondeductible quotas

- Current command verification setting

For more information see the VAX/VMS Command Language User's Guide.

Format

ret-status = LIB$SPAWN(<command-string.rt.dx><, <input-file.rt.dx> <output-file.rt.dx><, <flags.rlu.r><, <process-name.rt.dx> <, <process-id.wlu.r><, <completion-status.wlc.r> <, <completion-efn.rbu.r><, <completion-astadr.szem.r> <, completion-astprm.rz.v>>>>>>>>>)

command-string

A CLI command to be executed by the spawned subprocess. If omitted, commands are taken from the file specified by input-file. See notes below for additional information. Passed by descriptor.

input-file

An equivalence name to be associated with the logical name SYS$INPUT in the logical name table for the subprocess. If omitted, the default is the caller's SYS$INPUT. See notes below for additional information. Passed by descriptor.

output-file

An equivalence name to be associated with the logical names SYS$OUTPUT and SYS$ERROR in the logical name table for the subprocess. If omitted, the default is the caller's SYS$OUTPUT. Passed by descriptor.

flags

A longword of flag bits designating optional behavior. If omitted, the default is that all flags are clear. Passed by reference. The flags defined are:

Bits 0 NOWAIT

If set, the calling process continues executing in parallel with the subprocess. If clear, the calling process hibernates until the subprocess completes.

Bit 1 NOCLISYM

If set, the spawned subprocess does not inherit CLI symbols from its caller. If clear, the subprocess inherits all currently defined process logical names. You may want to specify NOLOGNAM to help prevent commands redefined by logical name assignments from affecting the spawned commands.

Bits 3 through 31 are reserved for future expansion and must be zero.

<u>process-name</u>

The name desired for the subprocess. If omitted, a unique process name will be generated. Passed by descriptor.

<u>process-id</u>

The longword to receive the process identification of the spawned subprocess. This value is only meaningful if the NOWAIT <u>flags</u> bit is set. Passed by reference.

<u>completion-status</u>

The longword to receive the subprocess' final completion status. If the NOWAIT <u>flags</u> bit is set, this value is not stored until the subprocess completes; use the <u>completion-efn</u> or <u>completion-astadr</u> parameters to determine when the subprocess has completed. Passed by reference.

<u>completion-efn</u>

The number of a local event flag to be set when the spawned subprocess completes. If omitted, no event flag is set. Specifying this parameter is only meaningful if the NOWAIT <u>flags</u> bit is set. Passed by reference.

<u>completion-astadr</u>

The entry mask of a procedure to be called by means of an AST when the subprocess completes. Specifying this parameter is only meaningful if the NOWAIT <u>flags</u> bit is set and if <u>completion-astadr has been specified</u>.

<u>completion-astprm</u>

A value to be passed to the procedure specified by <u>completion-astadr</u> as an AST routine parameter. Typically, this would be the address of a block of storage to be read or written by the AST procedure. Specifying this parameter

is only meaningful if the NOWAIT _flags_ bit is set and in _Completion-astadr_ has been specified.

Notes

If neither _command-string_ nor _input-file_ is present, command input will be taken form the parent terminal. If both _command-string_ and _input-file_ are present, the subprocess will first execute _command-string_ and then read from _input-file_. If only _command-string_ is specified, the command will be executed and the subprocess will be terminated. If _input-file_ is specified, the subprocess will be terminated either by a LOGOUT command or an end-of-file.

The subprocess does not inherit process-permanent files, nor procedure or image context. No LOGIN.COM file is executed.

Unless the NOWAIT _flags_ bit is set, the caller's process is put into hibernation until the subprocess completes. Because the caller's process hibernates in supervisor mode, any user-mode ASTs queued for delivery to the caller will not be delivered until the caller reawakes. Control can also be restored to the caller by means of an ATTACH command or a suitable call to LIB$ATTACH from the subprocess.

This procedure is supported for use with the DCL command language interpreter. If used when the current CLI is MCR, the error status LIB$_NOCLI will be returned.

If an image is run directly as a subprocess or as a detached process, there is no CLI present to perform this function. In such cases, the error status LIB$_NOCLI is returned.

LIB$STOP- Stop Execution via Signaling

LIB$STOP is called whenever your program must indicate an exception condition or output a message because it

is impossible to continue execution or return a status code to the calling program. LIB$STOP scans the stack frame by frame, starting with the most recent frame, calling each established handler (see the <u>VAX-11 Run-Time Library User's Guide</u>). LIB$STOP guarantees that control will not return to the caller.

<u>Format</u>

CALL LIB$STOP (condition-value.rlc.r<,parameters.rl.v,...>)

<u>condition-value</u>

A standard signal for a VAX-11 32-bit condition value. Passed by immediate value.

<u>parameters</u>

Additional FAO parameters for message. Passed by immediate value. See the <u>VAX-11 Run-Time Library User's Guide</u> for the message format.

<u>Notes</u>

The argument list is copied to the signal argument list vector, and the PC and PSL of the caller are appended to the signal vector.

The severity of <u>condition-value</u> is forced to SEVERE before each call to a handler.

If any handler attempts to continue by returning a success completion code, the error message ATTEMPT TO CONTINUE FROM STOP is printed and your program exists.

If a handler calls SYS$UNWIND, control will not return to the caller, thus changing the program flow. A handler can also modify the saved copy of R0/R1 in the mechanism vector.

The only way a handler can prevent the image from exiting after a call to LIB$STOP is to unwind the stack using the SYS$UNWIND system service.

APPENDIX B

ETHERNET LOCAL AREA NETWORK

A convenient method of connecting computers over short distances is the Ethernet local area computer network. In fact, Ethernet has now been recognized by more than a dozen manufacturers as the de facto standard for local area computer communications.

The 10 Mbit per second, packet switching network is designed to interconnect hundreds of high-function computers or workstations within 2.5 kilometers of each other. Ethernet uses a passive, equitable, highly efficient statistical method known as carrier-sense multiple-access with collision detection (CSMA/CD) that enables stations on the network to share access to a 50-ohm coaxial cable transmission medium. A cable segmentcan be up to 500 m long and connect up to 100 stations. Each station attaches to a coaxial cable via a tranceiver system, through a cable that connects the tranceiver to the station and can not exceed 50 m in length.

Messages are formatted into standard frames made up of bytes. Framing consists of a destination portion (6 bytes), a source portion (6 bytes), the message type (2 bytes), data (46 to 1500 bytes), and a frame-check sequence (4 bytes). Messages can be addressed to a single station, to all stations (broadcast), or to a number of selected stations. Signals are transmitted using Manchester encoding, a means of combining separate data and clock signals into a single, self-synchronizable data stream suitable for transmission on a serial channel.

The CSMA/CD approach can be summarized as follows:

Carrier-sense means that each station "listens" to the cable before transmitting a packet; if some other station is already transmitting, the first station senses the presence of a carrier and defers transmitting its own packet until the cable is quiescent.

Multiple-access means that all stations tap into and share the same coaxial cable. Every transmitted packet is "heard" by all stations on the Ethernet. The intended recipients detect incoming packets by recognizing their addresses embedded in the packets; other packets are discarded.

If two or more stations transmit packets at the same time, their signals will be intermixed on the coaxial cable. This is known as a collision. By listening while transmitting and comparing what is heard on the cable with the data being transmitted, each station can detect collisions and back off by waiting a random time interval before attempting to retransmit the packet. The efficiency of the network remains high even under conditions of heavy load, because the mean of the random back off interval increases each time a collision occurs.

Related Information

In the Figure B.1 is depicted in steps the decision of selecting which medium to pick for a local network. A user can move from left to right accross the selection "tree", checking the distances, bandwidth, and applications supported by twisted pair, baseband, and broadband medium classes. Optical fiber currently seems best only for point-to-point communications.

Like the physical medium, choises are available for the access method. Figure B.2 is a selection tree to determine the optimal access method for a specific operating environment. So, a prospective user can again can move from left to right to see the transmission characteristics supplied by the three most popular access techniques:

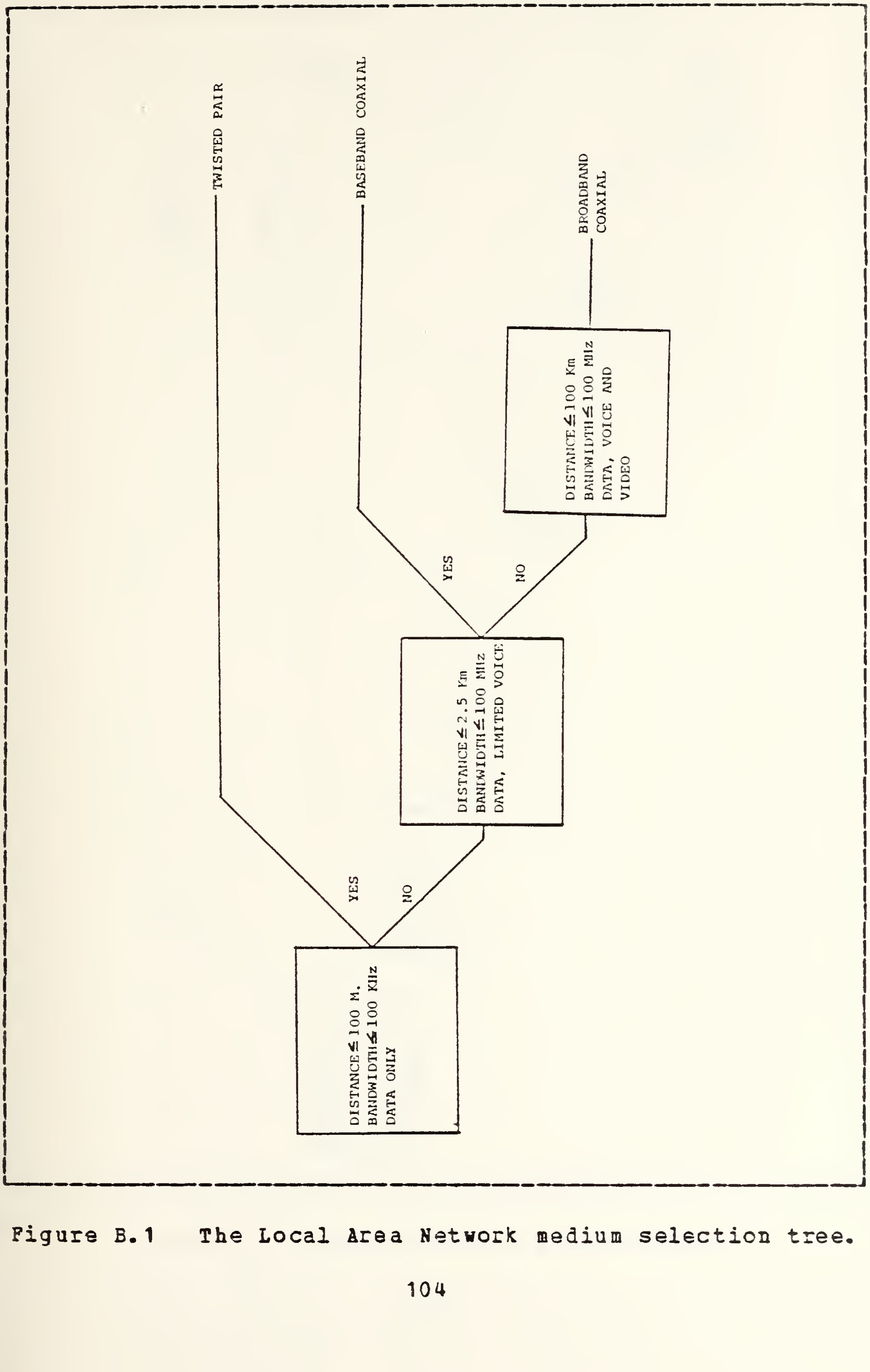

Figure B.1 The Local Area Network medium selection tree.

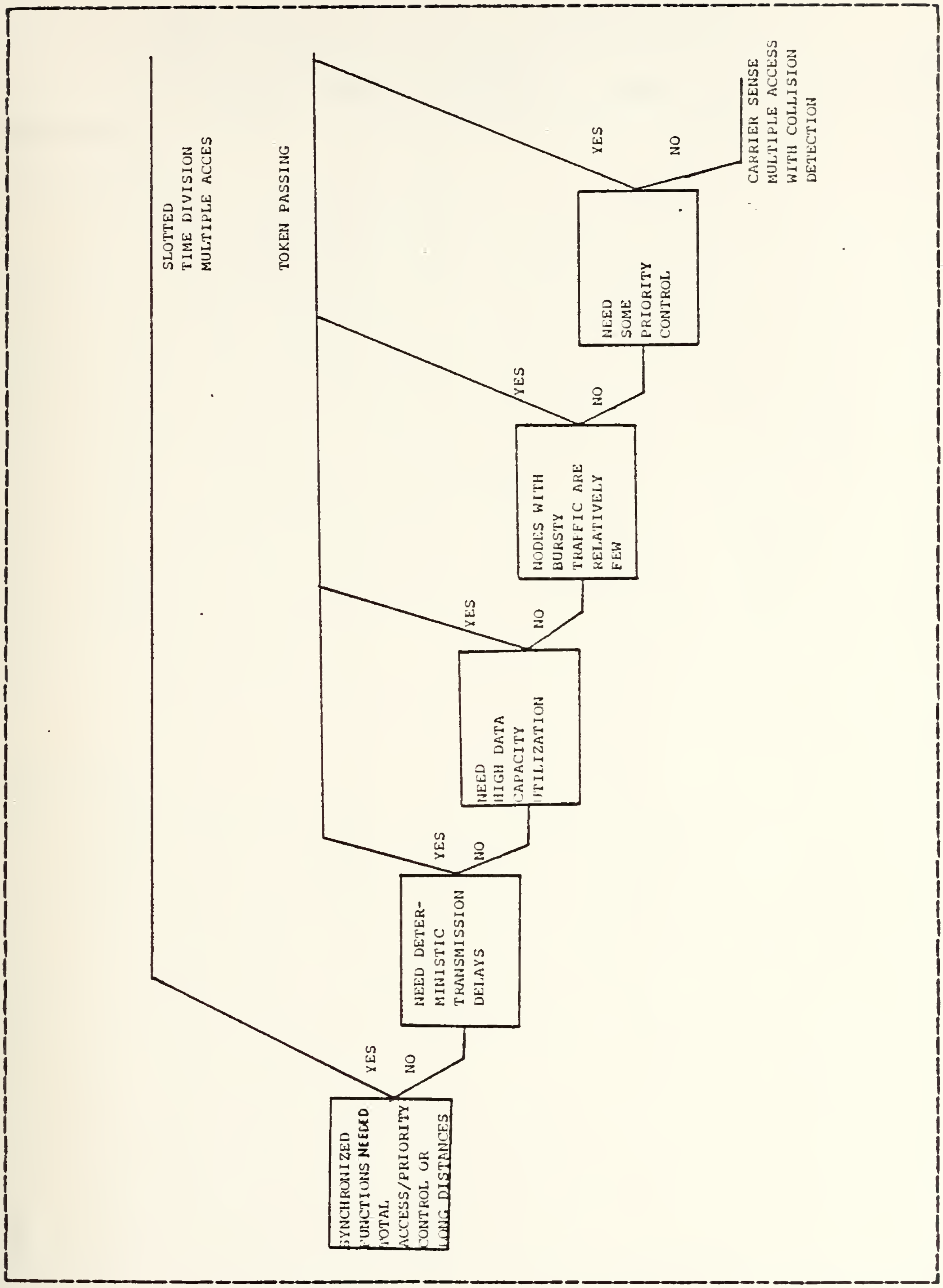

Figure B.2 Selecting the access method.

- Slotted time division multiple access
- Token passing
- Carrier sense multiple access with collision detection

APPENDIX C

NI1010 BOARD. DESCRIPTION-FEATURES

DESCRIPTION

The NI1010 UNIBUS Ethernet Communications Controller is a single hex-height board that contains all the data communications controller logic required for interfacing DEC's family VAX-11 and PDP-11 minicomputers to an Ethernet local area network. THe controller board complies in full with the Xerox/Intel/Digital Ethernet V1.0 specification by performing the specified CSMA/CD data link and physical channel functions. Also, when attached to a tranceiver unit, provides the host UNIBUS system a complete connection onto the Ethernet basebandcoaxial cable local area network, permitting 10 Mbit per second transmissions over distances up to 2500 meters.

FEATURES

Implements Ethernet Data Link Layer Functions

The NI1010 formats frames and performs the CSMA/CD transmit link management functions required to successfully deliver frames onto the network. When not transmitting a frame, the NI1010 continuously listens to the network for frame traffic intended for it. Only frames with a matching address are accepted by the controller for subsequent transfer to the host UNIBUS system. The controller performs Physical, Multicast-Group (up to 63), and Broadcast address recognition. CRC generation and checking is also performed.

<u>Implements Ethernet Physical Channel Functions</u> The NI1010 transmits and receives 10 Mbits per second bit streams with

electrical and timing specifications compatible for direct connection to an Ethernet tranceiver unit. The controller performs the required frame synchronization functions, and Manchester encoding/decoding of the bit streams.

Supports High Station Performance

The NI1010 has being designed to offer high network performance while minimizing the service load placed upon the host UNIBUS system. Serving to buffer the host from the unpredictable interarrival times characteristic of network traffic, the board has a receive FIFO (first-in, first-out) memory which can store up to 13.5 Kbytes of received frames. For transmit buffering, the NI1010 has a 1.5 Kbyte FIFO from which all frame retransmissions are made. All data transfers between the NI1010 and host UNIBUS memory are performed by the NI1010's DMA controller. The DMA controller may be preloaded by the host with up to 16 receive buffer descriptors.

Extensive Diagnostic Features :

The NI1010A controller offers comprehensive network and board -level diagnostic capabilities. LED indicators mounted on the edge of the board provide a visual indication of whether or not the host is communicating onto the network. For a comprehensive station diagnosis, the NI1010A may be operated in three different types o data loopback. On power-up, or by host command, the controller performs a confidence test on itself. A LED indicator showes the pass/fail operational state of the board.

Collects Network Statistics:

The NI1010A tallies statistical values on various network trafficand error conditions observed.

One Hex-Height UNIBUS Board:

The NI1010A fits into one UNIBUS SPC slot. The controller is mechanically, electrically, and architecturally compatible with Digital Equipment Corporation's UNIBUS specifications.

SPECIFICATIONS

- 10 million bit per second data transmission rate

- Coaxial cable segments up to 500 meters (1640 feet) in length.

- Up to 100 transceiver connections per coaxial cable segment.

- Up to 2 repeaters in path between any two stations.

- Up to 1500 meters (4920 feet) of coaxial cable between any two sections.

- Up to 50 meters (165 feet) of transceiver cable between an NI1010A controller and its transceiver.

- Up to 2500 meters (1.55 miles) maximum station separation.

- Point-to-point links up to 1000 meters (3280 feet) in length.

-Up to 1024 stations per network.

APPENDIX D

NI1010 ETHERNET CONTROLLER MULTIPLEXING USER'S MANUAL

A. GENERAL INFORMATION

"Ethermult" and "Usermult" are two programs that provide the means for the multiplexing of the NI1010 Ethernet Communications Controller Board which takes the role of the interface between VAX-11/780 and Ethernet. In their present form, they enable nine users to access the VAX/VMS facilities from an MDS terminal provided that:

1. The program "Ethermult" is running in a VAX/VMS terminal.

2. Each user has his own "Usermult" program running in a VAX/VMS terminal.

In this scheme, each user can execute VMS commands from his terminal as if he had a real VAX/VMS terminal to do his job.

B. SPECIFIC INFORMATION

Both programs reside on the VAX/VMS under the public user account with user name "INTERLAN" and password "VMS" .

First thing that a person willing to work with the multiplexing should have, is an account in VAX/VMS. This can be easily arranged through Mrs Olive M. Paek of the VAX-11 professional staff (Rm 525B).

Next he should login in a VAX/VMS terminal and type the following commands:

```
$COPY  <CR>
$FROM:  _DRA1: INTERLAN ETHERMULT.EXE  <CR>
$TO:     *  <CR>
```

The same commands should be repeated for the "USERMULT.EXE" file.

Now, the "Ethermult" should be executed from a VMS terminal and after that, the "Usermult" program should also be executed in as many VMS terminals as many users are required. This can be done by typing:

$ R ETHERMULT <CR>

$ R USERMULT <CR>

All programs, as they are set up now, must be executed from one directory i.e. the login procedure at each VMS terminal should be done using the same user name and password. This happens since the main program ("Ethermult") can only have access in the answer files that reside on its directory. That is, the answer file of the user, say, 9 (REPL9.DAT) should be in the same directory the ETHERMULT.EXE is executing. Otherwise, when the program tries to find it in order to send it to MDS it will fail since they reside in different directories.

1. <u>Operation on MDS Systems</u>

After the required number of "USERMULT" programs along with the "ETHERMULT" have been executed in different VMS terminals, the modified "ETHERNET" program (see Appendix E) must be executed in as many MDS terminals (currently only two are available) as the number of users is.

There are two diskettes with the same "ETHERNET" program, one for the single and one for the double density MDS. The procedure in the MDS side, same for either of them, is the following:

When the system is booted up with the corresponding diskette, execute ETHERNET.COM by typing:

A> ETHERNET <CR>

Now , a series of prompts appears on the MDS screen:

```
****************************************************
ETHERNET COMMUNICATION PROGRAM-VERSION 5.0
ALLOWS THIS HOST TO CONNECT TO THE NET.
CNTL-H=BACKSPACE FOR TEXT ENTRIES:
****************************************************
```

```
************ MAIN MENU ************
WRITE RECEIVED FILES TO DISK NO:
DEFAULT DRIVE(A)    = 1
DISK DRIVE A        = 2
DISK DRIVE B        = 3
*********************************************
ENTER DRIVE NUMBER==>
```

The user can enter the drive number he wishes without affecting the program, since no data is going to be transfered to the MDS diskettes. After typing in one of the three numbers (1,2 or 3) another set of prompts appears:

```
ETHERNET FRAME DATA BLOCK SIZE:
SELECT 128 FOR ALL FILE OPERATIONS
AND VAX COMMUNICATIONS.
    128 BYTES    = 1
    256 BYTES    = 2
    512 BYTES    = 3
   1024 BYTES    = 4
   1500 BYTES    = 5
*************************************
ENTER SELECTION==>
```

Here also any selection will not affect the program since the frame size is fixed to 1500 bytes when the virtual terminal mode (nr. 3) is selected in the next set of prompts:

```
OPERATING MODES:
************************************
RECEIVE WAIT LOOP                 = 1
TRANSMIT FILE OR MESSAGE          = 2
VIRTUAL TERMINAL OF VAX           = 3
VAX COMMAND MODE                  = 4
DISCONNECT FROM NET               = 5
************************************
ENTER SELECTION==>
```

Now the "VIRTUAL TERMINAL OF VAX" (nr. 3) <u>must be selected</u>.

After this, a "V>" appears and the user is ready to type his own commands. They should be the usual VMS commands, preceded by the characteristic number of the user, selected when the "Usermult" program was executed.

Example: V>1dir <CR> (for the user nr 1)

or V>3show time <CR> (for the user nr 3)

When the answer to the command appears on the screen, a new command can be typed in <u>after the "V>" reappears</u>.

If the user wants to finish his session, he can type a "." <CR> and the sets of selection prompts appear again. If he wants to logout of the VAX/VMS multiplexing system he should type "LO" and the "Usermult" corresponding to him will exit. From then on, this user cannot enter the

multiplexing system except if he runs again the "Usermult" program from a VMS terminal.

APPENDIX E

```
C
C       AUTHORS: MAJOR ANTONIOS SAKELLAROPOULOS
C                      HELLENIC AIR FORCE
C
C                 LIEUTENANT IOANNIS KIDONIEFS
C                      HELLENIC NAVY
C
C       THESIS ADVISOR: PROF. UNO KODRES
C
C       NAVAL POSTGRADUATE SCHOOL, DECEMBER 1983
C
C
C       This program performs the  multiplexing  of Ethernet Interface
C       among VAX users.  It should  run  in  conjuction  with program
C       "Usermult". Detailed description of both programs can be found
C       in the  thesis of  the authors, under the  title "Multiplexing
C       the Ethernet Interface Among VAX users".
C
C
C
                program ethermult

c       variables' declaration :

        implicit        integer*4(a-z)
        integer*2       iosb(2),c,condition
        integer*4       nichan, sys$qiow, sys$assign
        include         '←dra0:[npssys.interlan]nidef.for'
        include         '($iodef)'
        include         '($ssdef)'
        byte            MRpacket(1522),MTpacket(1508),usrnum,
        1               dflag,flag,com(81),
        2               table(9,4),row,rowflag
        character       alpha(9)/'1','2','3','4','5','6','7','8','9'/,
        1               mailbox*7/'usrmail'/,
        2               msg*27/'Invalid user #. Msg ignored'/,
        3               msg1*27/'Missing user #. Msg ignored'/
        external        dummy
        common          nichan/pak/mqueue(9,81)/fl/ackflag/slot/n,
        1               /ind1/notyet(9)/ind2/times(9)/filunit/unitnr


c     Initializations :

        c=0
        unitnr=0
        do i=1,9
           do j=1,4
              table(i,j)=0
           enddo
           do l=1,81
              mqueue(i,l)=0
           enddo
           notyet(I)=0
        enddo
        m=11
        do i = 1,9
           times(i) = m
```

```
            m = m + 1
         end do
C     Associate the common event flag cluster NET :
         status = sys$ascefc(%val(o4),'net',,)
         if (.not.status) call lib$stop(%val(status))

C     Initialize common event flags #64 to 84 to zero :
         do i=64,84
            status = sys$clref(%val(i))
            if (.not.status) call lib$stop(%val(status))
         end do

C     Assign a channel to NIA0:
         istat=sys$assign('NIA0',nichan,,)
         if(.not.istat) type *, ' Assign error!'
C     Start up and go on line:
         istat= sys$qiow(,%val(nichan),
         1                  %val(io$+setmode .or. io$m+startup),
         2                 iosb,,,,,,,,)
         if(.not.istat) type *,' Istat start up error!'
         if(iosb(1).lt.0) type *, ' Start up error!'

c     Initial setting of the receive-mode :
         call rec(MRpacket)       ! Receive the command with the user number.

10       i = 1
         condition = 0
         do while ((condition.eq.0).and.(i.le.9).and.(mqueue(i,7).ne.0))
            usrnum = mqueue(i,19)
            usrnum = usrnum - 48              ! "Convert" to decimal.
c     Check the user number :
            if (usrnum.eq.0) then             ! Invalid user number.
               call message(msg,mqueue(i,15),mqueue(i,16))
               do j = 1,81
                  mqueue(i,j) = 0             ! Clear this slot.
               enddo
               call arrqueue                  ! Fill up the emptied slot.
               go to 10
            else if (usrnum.gt.9) then        ! Missing user number.
               call message(msg1,mqueue(i,15),mqueue(i,16))
               do j = 1,81
                  mqueue(i,j) = 0
               enddo
               call arrqueue
               go to 10
            end if

            call search(usrnum,table,c,rowflag,row)
            if (table(row,2).eq.1) then   ! Reply file for previous command
                                          ! of this user hasn't been sent yet
               i = i + 1                  ! serve next user.

            else
               condition = 1

c     Extract the command and put it in a buffer :
               do j=1,81
                  com(j)=mqueue(i,j) ! Load the buffer with the command.
                  mqueue(i,j)=0      ! Zero the command buffer.
               enddo
               call arrqueue
```

```
c    Write the command to the user"s mailbox :
                call distrioution(com,usrnum,dflaq,table,msq,c)

                if (dflag.eq.1) go to 10   ! No such user number finally.
             end if
          enddo

c    Send the command to the user :
          call exoort(c,table,MRoacket)

c    Check the status of the users and update user table and number
c    of current users, if neccesary.
          call status-check(c,table)

          go to 10

20        end

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

          subroutine SENDMSG(outfile,ans,usernum,MRoacket,c,table)
C
C     ACTUAL SENDING OF MESSAGE.
C     WAIT FOR ACKNOWLEDGE.
C     IF NOT ACKNOWLEDGE IN 5 SEC, RETRANSMIT.
C     IF NOT ACKNOWLEDGE IN 10 SEC, TRANSMISSION.

          implicit          integer*4(a-z)
          integer*2         iosb(2),endfil,nchars,last /0/,c
          integer*4         nichan, sys$qiow, sys$assign
          include           '←dra0:[ncssys.interlan]nidef.for'
          include           '($iodef)'
          include           '($ssdef)'
          byte              Temobuff(80),ans(2),usernum,MRoacket(1522)
          byte              STpacket(1508),SRcacket(1522),row,
          1                 table(9,4),rowflag
          character         outfile*9,del*6/'$ del '/,vers*2/';*'/,
          1                 delfile*17,
          2                 alpha(9)/'1','2','3','4','5','6','7','8','9'/
          common            nichan/fl/ackflag/ind1/notyet(9)/ind2/times(9),
          1                 /filunit/unitnr


          delfile=del//outfile//vers

C     Assign destination address:
          STpacket(1)='02'x
          STpacket(2)='07'x
          STpacket(3)='01'x
          STpacket(4)='00'x
          STpacket(5)=ans(1)
          STpacket(6)=ans(2)
          STpacket(7)='00'x
          STpacket(8)='00'x


          endfil=0
20        do i=9,1508
```

```
                STpacket(i)='00'x
        enddo

        j=9
        do while(j.le.1425)!store no more than 1500 characters in STpacket
                read(times(unitnr),21,end=40) nchars,(Tempbuff(i),i=1,80)
21              format(q,80a1) ! count the nr of characters in the line
22              do m=1,nchars
                        STpacket(j)=Tempbuff(m)
                        j=j+1
                enddo
                STpacket(j)='0D'x          ! carriage return at end of line
                STpacket(j+1)='0A'x        ! line feed
                j=j+2

        enddo




28      call tranceive(STpacket)           ! send the packet
        status=sys$waitfr(%val(2))
        if (.not.status) call lib$stop(%val(status))

C     Check if an acknowledge was sent :
        if ((MRpacket(18).eq.'FF'x).or.(ackflag.eq.ss$_wasset)) then
                status=sys$clref(%val(2))
                if (.not.status) call lib$stop(%val(status))
                if(endfil.ne.-1) GO TO 50  ! Send only one frame of this file
                go to 47
        end if


40      close(unit=times(unitnr))
        STpacket(8)='0F'x          ! For last packet recognition from MDS.
        endfil=-1                  ! Flag raised at end-of-file.
        go to 28                   ! Transmit the contents of the last packet.

47      status=lib$spawn(delfile)
        if (.not.status) call lib$stop(%val(status))
        status = sys$clref(%val(63+usernum))
        if (.not.status) call lib$stop(%val(status))

        call search(usernum,table,c,rowflag,row)
        if (rowflag.eq.1) then
           table(row,2) = 0 ! User free to enter distribution if needs so.
        end if

50      if(endfil.ne.-1) then
          notyet(usernum) = 1
        else
          notyet(usernum) = 0
        end if
        return
        end

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC


        subroutine tranceive(Tpack)
```

```
      implicit            integer*4(a-z)
      integer*2           ioso(2)
      integer*4           nichan
      include             '←dra0:[nossys.interlan]nidef.for'
      include             '($iodef)'
      include             '($ssdef)'
      byte                Toack(1508)
      common              nichan

c   Load transmit data and send:
      istat=sys$qiow(,%val(nichan),
      1                   %val(io←←ltds),
      2                   iosb,,,Toack,%val(1508),,,,)

      if(iosb(1).lt.0) call lib$stop(%val(iosb(1)))
      if(iosb(2).ne.0) call lib$stop(%val(iosb(2)))

      return
      end


cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      subroutine distribution(com,usernum,dflag,table,msg,c)


c           This routine searches  the user  information  table to find out
c       whether a given number exists as a valid  user number. If it is so
c       it puts the message into a mailbox that is indexed with the number
c       of the user to whom the msg was addressed.
c           If the given number was not a valid user number, the message is
c       ignored. In this case the "dflag" is returned with value "1".


      implicit            integer*4(a-z)
      include             '←dra0:[nossys.interlan]nidef.for'
      include             '($iodef)'
      include             '($ssdef)'
      byte                dflag,com(81),table(9,4),usernum,row,
      1                   rowflag
      character           alpha(9)/'1','2','3','4','5','6','7','8','9'/,
      1                   mailbox*7/'usrmail'/,msg*27
      character           outfile*9,out*4/'repl'/,fil*4/'.dat'/
      integer*2           c,channel(9)

      call search(usernum,table,c,rowflag,row)
      IF ((rowflag.eq.1).and.(table(row,2).eq.1)) then ! User number exists
c                                in user table and he has answer in process.
         return
      ELSE

         i = 1
         dflag = 0
         outfile=out//alpha(usernum)//fil
c    Check if the user number exists in the user info table :
         if(c.gt.0) then
            do while ((i.le.c).and.(table(i,1).ne.usernum))
               i = i+1
            end do
```

```
            if (i.gt.c) then ! The msg is addressed to an unidentified user
6              status = sys$readef(%val(72+usernum),usr) ! Check if the
c                                                  corresponding flag is set
               if (.not.status) call lib$stop(%val(status))
               if (status.ne.ss$←wasset) then ! Not a valid user finally.
                  dflag = 1
                  call message(msg,com(15),com(16))
                  return
               else ! Valid user, update user table
                  c = c + 1
                  table(c,1) = usernum
                  table(c,2) = 1 ! This user has passed through distributio
                  table(c,3) = com(15)  ! Associate addresses
                  table(c,4) = com(16)
               end if
            else ! This user already is authorized to use the system.
               table(row,2) = 1 ! Indicate pass through distribution.
               table(row,3) = com(15) ! Associate addresses.
               table(row,4) = com(16)
            end if
         else
            go to 6
         end if
c    ***********************************************************************
c     create mailbox and assign a channel to it :
         status=sys$crembx(,channel(usernum),,,,,mailbox//alpha(usernum))
         if (.not.status) then
              type *,'error in creating user mailbox'
              call lib$stop(%val(status))
         endif

c     Write the command to user mailbox :
         status=sys$qiow(,%val(channel(usernum)),%val(io←←ltds),,,,
     1                %ref(com),%val(81),,,,)
         if (.not.status) then
              type *,'error in writing user"s mailbox'
              call lib$stop(%val(status))
         endif
      END IF

      return
      end


cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc


      subroutine export(c,table,MRpacket)



c             This routine  finds out  which  user has priority to send his
c     reply  to NI1010  controller and  sends one frame (1500 bytes) of
c     it. Then  proceeds to the  next ready answer, sends one frame and
c     so on, until all users with ready answer have send one frame.


      implicit        integer*4(a-z)
      include         '($ssdef)'
      integer*2       c,i
```

```
      byte              addr(2),table(9,4),usernum,MRpacket(1522)
      character         alpha(9)/'1','2','3','4','5','6','7','8','9'/,
      1                 outfile*9,out*4/'repl'/,fil*4/'.dat'/
      common/ind1/notyet(9)/ind2/times(9)/filunit/unitnr



      if (c.eq.0) return

      do i = 1,c
         usernum = table(i,1)
         outfile = out//alpha(usernum)//fil
         m = 1
         do while(outfile(5:5).ne.alpha(m))
            m = m + 1
         end do
         unitnr = m
         status = sys$readef(%val(63+usernum),usr)
         if (.not.status) call lib$stop(%val(status))
         if ((status.eq.ss$_wasset).and.(notyet(usernum).eq.0)) then
            open(unit=times(unitnr),file=outfile,status='old')
         end if
         if (status.eq.ss$_wasset) then ! There is an answer.
            addr(1) = table(i,3) ! Form the  address of
            addr(2) = table(i,4) ! the current user.
            outfile = out//alpha(usernum)//fil
            call sendmsg(outfile,addr,usernum,MRpacket,c,table)
         end if
      end do

      return
      end

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC


      subroutine message(msg,a1,a2)


      implicit          integer*4(a-z)
      character*27      msg
      byte              a1,a2,Tpack(1508)


      Tpack(1) = '02'x
      Tpack(2) = '07'x
      Tpack(3) = '01'x
      Tpack(4) = '00'x
      Tpack(5) = a1
      Tpack(6) = a2
      Tpack(7) = '00'x
      Tpack(8) = '0F'x

      k=9
      do i=1,27
              Tpack(k)=ichar(msg(i:i))
              k=k+1
      enddo
      Tpack(k+1)='0D'x
      Tpack(k+2)='0A'x
```

```
      Tpack(k+3)='20'x
      Tpack(k+4)='20'x

      call tranceive(Tpack) ! send msg and receive acknowlege

      return
      end

cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc


      subroutine xmit (Tpack,MRpacket)

c     This subroutine transmits an already formed packet


      implicit        integer*4(a-z)
      integer*2       iosb(2)
      integer*4       nichan
      include         '←dra0:[nossys.interlan]nidef.for'
      include         '($iodef)'
      byte            Tpack(1508),MRpacket(1522)
      common          nichan/pak/mqueue(9,81)


      Tpack(1)='02'x
      Tpack(2)='07'x
      Tpack(3)='01'x
      Tpack(4)=MRpacket(14)
      Tpack(5)=MRpacket(15)
      Tpack(6)=MRpacket(16)
      Tpack(7)='00'x
      Tpack(8)='FF'x
      Tpack(9)='60'x
c     Load transmit data and send:
      istat=sys$qiow(,%val(nichan),%val(io←←ltds),
     1                iosb,,,Tpack,%val(1508),,,,)

      if    (iosb(1).lt.0) then
              type *,' Ether xmit error!!'
              call lib$stop(%val(iosb(1)))
      else if(iosb(2).ne.0) then
              type *,' Controller xmit error!!'
              call lib$stop(%val(iosb(2)))
      else
              i=1       ! dummy
      endif

      return
      end


cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc


      subroutine dummy(MRpacket)
```

```
c       This is the routine in which control of the program is transferred
c       when a message arrives at the NI1010 board and an AST occurs.


        implicit          integer*4(a-z)
        include           '($iodef)'
        include           '($ssdef)'
        byte              MRpacket(1522),Topack(1508),com(81)
        common/oak/mqueue(9,81)/fl/ackflag/slot/n


        if (MRpacket(18).eq.'FF'x) then   ! Acknowledge packet received.
           call rec(MRpacket)             ! Reset the receive mode.
           ackflag=sys$setef(%val(2))     ! Set the acknowledge flag.
           if(.not.ackflag) call lib$stop(%val(ackflag))

        else                              ! Command packet received.
           call xmit(Topack,MRpacket)     ! Send acknowledge to MDS.

c   Extract the command from the received packet :
           i=19
           do while (MRpacket(i).ne.ichar('`'))
              i=i+1
           enddo
           k=1
           do j=1,i
              com(k)=MRpacket(j)
              k=k+1
           enddo
           call arrqueue       ! Arrange the command queue.
           call formqueue(com) ! Put the command at the first
                                ! empty slot of the queue.
           call rec(MRpacket)   ! Reset the receive mode.
        end if
        return
        end

cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc



           subroutine rec(MRpacket)

c             This subroutine receives a packet from MDS. This packet can be
c      either a command packet (18th byte = 00) or an acknowledge packet
c      (18th byte = FF). When it is called, it sets up the NI1010 receive
c      mode and exits. Then, as soon as a packet arrives at the NI1010 it
c      interrupts the current flow of the program (AST is "triggered") and
c      calls the AST subroutine DUMMY.


        implicit          integer*4(a-z)
        integer*2         iosb(2)
        integer*4         nichan
        include           '←dra0:[nossys.interlan]nidef.for'
        include           '($iodef)'
        byte              MRpacket(1522),Topack(1508)
        common            nichan/oak/mqueue(9,81)
        external          dummy
```

```
      status=sys$clref(%val(2))
      if (.not.status) call lib$stop(%val(status))
c     type *,'Ready to receive....',nichan
      istat=sys$qio(%val(1),%val(nichan),%val(io$+readlblk),
     1     .           iosb,dummy,MPpacket,MPpacket,%val(1522),,,,)

      if (iosb(1).lt.0) then
              type *,' Ether error in reception of msg in VAX/VMS!!'
              call lib$stop(%val(iosb(1)))
      else if (iosb(2).ne.0) then
              type *,' Rcv error in VAX/VMS Ethernet controller !!'
              call lib$stop(%val(iosb(2)))
      endif

      return
      end

cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc


      subroutine formqueue(com)

c          This subroutine puts the new command into the first empty slot
c     of the command queue.

      implicit          integer*4(a-z)
      byte              com(81),MPpacket(1522)
      common/pak/mqueue(9,81)/slot/n

      do j=1,81
         mqueue(n,j)=com(j) ! Fill up the first empty slot of the queue.
      enddo
      return
      end

ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc


      subroutine arrqueue

      implicit          integer*4(a-z)
      common/pak/mqueue(9,81)/slot/n

c     Arrange the queue :
      m=1
      do while (m.le.9)
         if((mqueue(m,7).eq.0).and.(mqueue((m+1),7).eq.1)) then
            do i = 1,81
               mqueue(m,i) = mqueue((m+1),i)
               mqueue((m+1),i) = 0
            enddo
         end if
         m = m + 1
      enddo

c     Locate the first empty slot in the queue :
      n=10
      do i=9,1,-1
         if(mqueue(i,7).eq.0) then
```

```
            n=n-1
         end if
      enddo
      if(n.eq.10) type *,'*** Queue is full. Command not queued !! ***'
      return
      end

cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      Subroutine status_check(c,table)

c        This routine  reads the flags of current users to check if they
c     are still in the system. If a flag was found reset, that means the
c     corresponding user has logged out. Then the user  table and number
c     of users in  the system (c)  are updated. The user  table is  then
c     rearranged.

      implicit          integer*4(a-z)
      integer*2         change,i,c,k
      byte              table(9,4),user
      include           '($ssdef)'
      include           '($iodef)'

      change = 0
      k      = c

      if (k.gt.0) then
         do i = 1,k
            user = table(i,1)
c           Check whether this user is still in the system :
            status = sys$readef(%val(72+user),usr)
            if (.not.status) call lib$stop(%val(status))
            if (status.ne.ss$_wasset) then ! This user has logged out.
               change = 1 ! At least one change has occured.
               c = c - 1
               table(i,1) = 0
            end if
         end do
         if(change.eq.1) call arrange(k,table)
      end if

      return
      end

cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      Subroutine arrange(k,table)

c        This routine rearranges the user table after at least one user has
c     left the system, so that there are no empty lines between full ones.

      integer*2         m,i,k,j
      byte              table(9,4),temp(9,4)

      m = 1
      i = 1
      do while(i.le.k)
```

```
         if (table(i,1).ne.0) then
            do j = 1,4
               temp(m,j) = table(i,j)
               table(i,j) = 0
            end do
            i = i+1
            m = m+1
         else
            i = i+1
         end if
      end do

      do i = 1,m-1
         do j = 1,4
            table(i,j) = temp(i,j)
         end do
      end do

      return
      end

cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      subroutine search(usernum,table,c,rowflag,row)

c       This routine  searches the user table to find a specific usernum
c   which is given. If it finds it there it  returns  the "rowflag" with
c   value 1, and the "row" of the table in which this usernum was found.
c   If the  usernum was  not  found  there, the rowflag is returned with
c   value 0.

      integer*2         c,i
      byte              usernum,row,rowflag,
      1                 table(9,4)

      rowflag = 0
      row     = 0
      i       = 1

      do while((i.le.c).and.(usernum.ne.table(i,1)))
         i = i+1
      end do
      if(i.le.c) then
         row     = table(i,1)
         rowflag = 1
      end if

      return
      end
```

APPENDIX F

```
C
C       AUTHORS: MAJOR ANTONIOS SAKELLAROPOULOS
C                      HELLENIC AIR FORCE
C
C                 LIEUTENANT IOANNIS KIDONIEFS
C                      HELLENIC NAVY
C
C       THESIS ADVISOR: PROF. UNO KODRES
C
C       NAVAL POSTGRADUATE SCHOOL, DECEMBER 1983
C
C
C       This program should run together with program "ETHERMULT" in order
C       to achieve the Ethernet Interface multiplexing among VAX users.
C       Detailed description of the program is found in the thesis of the
C       authors under the same title.
C
C

            program usermult

      implicit          integer*4(a-z)
      integer*2         channel(9),iosb(2),endfil,doneflag,
      1                 load
      byte              com(81),ans(2),usernum
      include           '←dra0:[nossys.interlan]nidef.for'
      include           '($iodef)'
      character         mail*7/'usrmail'/,
      2                 alpha(9)/'1','2','3','4','5','6','7','8','9'/
      common            nichan


      call authorize(usernum,load) ! Get the usernum.
      if (load.eq.1) go to 101 ! System cannot accept new users.

c   create a user mailbox and assign a channel to it :
10    status=sys$crembx(,channel(usernum),,,,,mail//alpha(usernum))
      if(.not.status) then
            type *,'error in user mailbox creation'
            call lib$stop(%val(status))
      endif
c   read the mailbox :
      status=sys$qiow(,%val(channel(usernum)),%val(io$←readlblk),,,,
      1                 %ref(com),%val(81),,,,)
      if (.not.status) then
            type *,'read-user-mailbox error'
            call lib$stop(%val(status))
      endif


c    Associate common event flag cluster #2 with the name "NET" :
      status=sys$ascefc(%val(64),'net',,)
      if (.not.status) call lib$stop (%val(status))

      call feedfile(com)
      call spawn(usernum)
      status = sys$setef(%val(63+usernum))
      if (.not.status) call lib$stop(%val(status))
      go to 10
```

```
101     end

cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

        subroutine authorize(usernum,load)

c          This subroutine checks the common event flags 73 to 81 to
c       determine what user numbers are available for a new user. Then
c       it interacts with the user and acceots or rejects an entered
c       user number. If it acceots, the prooer common event flag is set.
c       If there are no available user numbers it returns the variable
c       "load" with value "1".

        implicit            integer*4(a-z)
        integer*2           load,i,cond,
        1                   iosb(2)
        integer*4           flagarray(9),J
        byte                usernum
        include             '($iodef)'
        include             '($ssdef)'
        include             '+dra0:[nossys.interlan]nidef.for'
        external            abort
        common/cl/cancel


c     Assign a channel to terminal :
        status = sys$assign('tt:',termchan,,)
        if (.not.status) call lib$stop(%val(status))

c     Start up and go on line :
        status = sys$qiow(,%val(termchan),
        1                   %val(io$+setmode.or.io$m+startup),
        2                   iosb,,,,,,,,)
        if (.not.status) call lib$stop(%val(status))

c     Initializations :
        do j=1,9
           flagarray(j) = 0
        end do
        counter = 0
        cancel  = 0
        load    = 0
        cond    = 0
        j       = 1
c     Associate common event flag cluster #2 under the name "NET" :
        status = sys$ascefc(%val(64),'net',,)
        if (.not.status) call lib$stop(%val(status))

c     Check if flag #84 is set (system occupied).
01      status = sys$readef(%val(84),state)
        if (.not.status) call lib$stop(%val(status))
        IF (status.eq.ss$+wasset) then ! System occupied.
           if (cond.eq.0) then
              type *,' Please wait, system occupied'
              cond = 1
              go to 01
           else
              go to 01
```

```
         end if
      ELSE ! System available.
         status = sys$setef(%val(84),'net',,) ! Set flag #84
         if (.not.status) call lib$stop(%val(status))

c     Check what event flags from 73 to 81 are set and put the remaining
c     ones in flagarray buffer :
         do i=1,9
            status = sys$readef(%val(72+i),usr)
            if (.not.status) call lib$stop(%val(status))
            if (status.ne.ss$_wasset) then ! This user number is available
               flagarray(j) = i
               j = j+1
            end if
         end do

         j = j-1
         if (j.eq.0) then
            type *,'System full!! No new users at the moment.'
            load = 1
            return
         else
9           type *,' You may choose one of the following'
            type *,' available user numbers.'
            do i = 1,j
               write(6,04)flagarray(i)
            end do
         end if
04    format(i4)

c     Set the timer for 10 seconds :
         call sys$bintim('0 ::10.0',systime) ! Convert 10 sec. to sys fmt.
         call sys$setimr(,systime,abort,)     ! Start the timer.When

c     Get the new user number :
         type *,' You have 10 sec to enter the user number :'
c     Read the user"s number :
         status = sys$qiow(,%val(termchan),%val(io$_readlblk),
     1                     iosb,,usernum,%val(1),,,,)
         if (.not.status) call lib$stop(%val(status))
         usernum = usernum - 48

c     Cancel the timer :
         call sys$cantim(,)

c     Check if the new user number is acceptable :
         i = 1
         do while((i.le.j).and.(flagarray(i).ne.usernum))
            i = i+1
         end do
         if ((i.eq.j+1).and.(counter.eq.0)) Then ! Wrong number entered.
            counter = counter + 1
            type *,' You have entered an illegal user number!!'
            go to 9
         else if ((i.eq.j+1).and.(counter.eq.1)) then
            cancel = 1
            call abort
         else ! Valid user number . Set the proper flag.
            status = sys$setef(%val(72+usernum),'net',,)
            if (.not.status) call lib$stop(%val(status))
            status = sys$clref(%val(84)) ! Reset flag #1.
```

```
            if (.not.status) call lib$stop(%val(status))
         end if
         type *,' User number accepted '
      END IF
      return
      end

cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc


      SUBROUTINE abort

      implicit          integer*4(a-z)
      common/cl/cancel


c     Associate common event flag cluster #2 with the name "NET" :
         status = sys$ascefc(%val(64),'net',,)
         if (.not.status) call lib$stop(%val(status))

c     Clear flag #84 :
         status = sys$clref(%val(84))
         if (.not.status) call lib$stop(%val(status))

         if (cancel.eq.1) then
            write(6,15)
15          format(' Second time illegal user number. Program aborted!!')
         else
            write(6,16)
         end if
16       format(' Time has expired. Restart the program.')

         call exit

         end


cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc


      SUBROUTINE feedfile(com)


      character*23      msg1/'Received successfuly.`'/
      integer*2         iosb(2),first
      integer*4         nichan, sys$qiow, sys$assign
      include           '+dra0:[nossys.interlan]nidef.for'
      include           '($iodef)'
      byte              RGpacket(1522),TGpacket(1508),ans(2),com(81)


      i=20
      do while (com(i).ne.ichar('`'))
             i = i+1
      end do
      open (unit=1,file='mail.com',status='old')

      write(6,11)(com(j),j=20,i-1)
```

```
        write(1,11)(com(j),j=20,i-1)
11      format(' ',<i>a1)
        close (unit=1)

        return
        end

cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc


        subroutine spawn(usernum)

c       This program spawns a  subprocess for  executing CLI commands.
c       The commands that are  going to be  executed are  contained in
c       the file 'Mail.com' which is the input of the run time routine
c       Lib$spawn. The results of the  execution are written in a file
c       called 'Repla.dat'.

        implicit            integer*4(a-z)
        character           file*15 /'mail.com'/,esc←null*2
        byte                esc←null←num(2) /'1b'x,'00'x/,usernum
        integer*2           file←len /8/,dflag
        equivalence         (esc←null,esc←null←num)
        external            ss$←notran
        character           outfile*9,string*40,out*4/'repl'/,fil*4/'.dat'/,
      1                     alpha(9)/'1','2','3','4','5','6','7','8','9'/
        character*15        filedit
        equivalence         (filedit,string(7:20))


        last = 0
        outfile=out//alpha(usernum)//fil
        dflag=0
        do while(status.ne.%loc(ss$←notran))
                status=sys$trnlog(file(1:file←len),file←len,file,,,)
        enddo

        if (file(1:2).eq. esc←null) then
                file(1:file←len) = file(5:file←len)
                file←len = file←len - 4
        endif


        open( unit=1,file='mail.com',status='old')
        read(1,6,end=7)string
6       format(a)
7       close(unit=1)
        if((string(2:5).eq.'edit').or.(string(2:5).eq.'EDIT')) then
                status= lib$spawn('$ edit'//filedit,,outfile)
                if(.not.status) call lib$stop(%val(status))
                go to 9
        end if
c     Check for a logout command.If so,reset user"s flag :
        if ((string(2:3).eq.'LO').or.(string(2:3).eq.'lo')) then
           status = sys$clref(%val(72+usernum))
           if (.not.status) call lib$stop(%val(status))
           last = 1
        end if
```

```
      status=lib$spawn(,file(1:file_len),outfile)
      if (.not.status) call lib$stop(%val(status))

9     type 10
10    format(' command done')
      if (last.eq.1) call exit ! The last command was logout.
      return
11    end
```

APPENDIX G

SOFTWARE PROTOCOL IN MDS USING ETHERNET LAN

The following programs, developed by Mark Stotzer [Ref. 2], provide the means of accessing the Ethernet via the NI3010 Ethernet Communications Controller. The same programs work in both MDS's presently available in NPS Spanagel Hall, rooms 523 and 525.

Two modifications were introduced in these programs in order to improve the efficiency and speed of VAX-MDS communication:

1. In the subroutine "Sendmsg" was added a new assignement, namely "TXBUFF(8) = 0" in two places in order to denote that the frame that is sent is a command and not an acknowlege. This was necessary to be done since, with the previous set up, the MDS was transmitting an acknowledge frame with the type field (bytes 8 and 9) of a command frame. So when the VAX was receiving an acknowlege, it was interpreted as a command frame causing communication problems.

2. In the subroutine "Conmsg" was done a transposition of the call statements to the subroutines "Emptbuf" and "Trmsg". This way when the MDS receives a frame it sends first the acknowlege frame to VAX and after that dumps it to console resulting in much faster exchange o f frames between MDS and VAX.

```
ETHERNET:/*MAIN MODULE-APPLICATION LAYER-ISO LEVEL 7*/

PROCEDURE OPTIONS (MAIN);

DECLARE
        /* LOCAL VARIABLES */
        COUNT7    FIXED BINARY(7),
        COUNT7A   FIXED BINARY(7),
        COUNT7B   FIXED BINARY(7),
        COUNT7C   FIXED BINARY(7),
        DSKNO     CHARACTER(1),
        FRAMD     CHARACTER(1),
        SELECT    CHARACTER(1),
        /* GLOBAL VARIABLES */
        RECFIL    FIXED BINARY(7) EXTERNAL,
        FRSIZE    FIXED BINARY(15) EXTERNAL,
        VTERM     FIXED BINARY(7) EXTERNAL,
        TRMODE    FIXED BINARY(7) EXTERNAL,
        /* GLOBAL DATA STRUCTURES */
        TXBUFF(1508) FIXED BINARY(7) EXTERNAL,
        RXBUFF(1522) FIXED BINARY(7) EXTERNAL,
        TXTBUF (128) FIXED BINARY(7) EXTERNAL,
        1 RXFCB EXTERNAL,
          2 DISK FIXED BINARY(7),
          2 FNAME CHARACTER(8),
          2 FTYPE CHARACTER(3),
          2 RFCB(24) FIXED BINARY(7).
        1 TXFCB EXTERNAL,
          2 DISK FIXED BINARY(7),
          2 FNAME CHARACTER(8),
          2 FTYPE CHARACTER(3),
          2 TFCB(24) FIXED BINARY(7),
        /* EXTERNAL MODULES */
        INIT      ENTRY,
        SENDATA   ENTRY,
        RECEIVE   ENTRY;

 /*LAST REVISION: 09/15/83-0900 ORIGINAL PROGRAM:07/29/83 */
 /*AUTHOR: CAPT. MARK D. STOTZER-USMC-AEGIS GROUP        */
 /*THESIS ADVISOR: PROFESSOR UNO R. KODRES-COMP. SCIENCE  */

PUT SKIP LIST('******************************************');
PUT SKIP LIST('ETHERNET COMMUNICATION PROGRAM-VERSION 5.0');
PUT SKIP LIST('ALLOWS THIS HOST TO CONNECT TO THE NET.');
PUT SKIP LIST('CNTL-H=BACKSPACE FOR TEXT ENTRIES:');
PUT SKIP LIST('******************************************');
PUT SKIP(2);
RECFIL=47;
COUNT7=1;
DO WHILE (COUNT7=1);
   COUNT7A=1;
   DO WHILE(COUNT7A=1);
      PUT SKIP(2);
```

```
    PUT SKIP LIST('************ MAIN MENU ************');
    PUT SKIP LIST('WRITE RECEIVED FILES TO DISK NO:');
    PUT SKIP LIST('DEFAULT DRIVE(A)    = 1');
    PUT SKIP LIST('DISK DRIVE A        = 2');
    PUT SKIP LIST('DISK DRIVE B        = 3');
    PUT SKIP LIST('**********************************');
    PUT SKIP LIST('ENTER DRIVE NUMBER==>');
    GET LIST(DSKNO);
    PUT SKIP(2);
    IF DSKNO='1' THEN
       DO;
         RXFCB.DISK=0;
         COUNT7A=2;
       END;
    ELSE
    IF DSKNO='2' THEN
       DO;
         RXFCB.DISK=1;
         COUNT7A=2;
       END;
    ELSE
    IF DSKNO='3' THEN
       DO;
         RXFCB.DISK=2;
         COUNT7A=2;
       END;
    ELSE
       PUT SKIP LIST('INVALID DRIVE NUMBER-REENTER:');
END;/*DO LOOP*/
COUNT7B=1;
DO WHILE (COUNT7B=1);
    PUT SKIP LIST('ETHERNET FRAME DATA BLOCK SIZE:');
    PUT SKIP LIST('SELECT 128 FOR ALL FILE OPERATIONS');
    PUT SKIP LIST('AND VAX COMMUNICATIONS.');
    PUT SKIP LIST('    128 BYTES    = 1');
    PUT SKIP LIST('    256 BYTES    = 2');
    PUT SKIP LIST('    512 BYTES    = 3');
    PUT SKIP LIST('   1024 BYTES    = 4');
    PUT SKIP LIST('   1500 BYTES    = 5');
    PUT SKIP LIST('******************************');
    PUT SKIP LIST('ENTER SELECTION==>');
    GET LIST(FRAMD);
    PUT SKIP(2);
    IF FRAMD='1' THEN
       DO;
         FRSIZE=128;
         COUNT7B=2;
       END;
    ELSE
    IF FRAMD='2' THEN
       DO;
         FRSIZE=256;
         COUNT7B=2;
       END;
    ELSE
```

```
   IF FRAMD='3' THEN
      DO;
        FRSIZE=512;
        COUNT7B=2;
      END;
   ELSE
   IF FRAMD='4' THEN
      DO;
        FRSIZE=1024;
        COUNT7B=2;
      END;
   ELSE
   IF FRAMD='5' THEN
      DO;
        FRSIZE=1500;
        COUNT7B=2;
      END;
   ELSE
      PUT SKIP LIST('INCORRECT CHOICE-REENTER:');
END;/* DO LOOP */
VTERM=0;
TRMODE=0;
CALL INIT;
PUT SKIP LIST('OPERATING MODES:');
PUT SKIP LIST('****************************');
PUT SKIP LIST('RECEIVE WAIT LOOP        = 1');
PUT SKIP LIST('TRANSMIT FILE OR MESSAGE= 2');
PUT SKIP LIST('VIRTUAL TERMINAL OF VAX = 3');
PUT SKIP LIST('VAX COMMAND MODE         = 4');
PUT SKIP LIST('DISCONNECT FROM NET      = 5');
PUT SKIP LIST('****************************************');
PUT SKIP LIST('ENTER SELECTION ==>');
GET LIST(SELECT);
PUT SKIP(2);
IF SELECT='1' THEN
   DO;
     TXBUFF(1)=2;
     TXBUFF(2)=7;
     TXBUFF(3)=1;
     PUT SKIP LIST('IN RECEIVE WAIT LOOP-TO RETURN TO');
     PUT SKIP LIST('MAIN MENU: ENTER <CR> ==>');
     PUT SKIP LIST('********************************');
     PUT SKIP(2);
     CALL RECEIVE;
   END;
ELSE
IF SELECT='2' THEN
   CALL TRANS2  ;
ELSE
IF SELECT='3' THEN
   DO;
     VTERM=1;
     FRSIZE=1500;
     PUT SKIP LIST('******* VAX TERMINAL MODE *******');
     PUT SKIP(1);
```

```
      PUT SKIP LIST('VAX TERMINAL SERVICE:');
      PUT SKIP LIST('DATA BLOCK SIZE PER FRAME=');
      PUT LIST(FRSIZE);
      PUT SKIP LIST('-----------------------------------');
      PUT SKIP LIST('TERMINAL ENTRY BY LINE OF TEXT');
      PUT SKIP LIST('BEGIN AFTER INITIAL V PROMPT: "V>"');
      PUT SKIP LIST('ENTER: TEXT LINE<CR>');
      PUT SKIP LIST('PROMPT WILL AUTOMATICALLY REAPPEAR');
      PUT SKIP LIST('UPON ENTRY OF THE FIRST CHARACTER');
      PUT SKIP LIST('OF THE NEXT LINE YOU BEGIN.');
      PUT SKIP LIST('-----------------------------------');
      PUT SKIP LIST('TO END TERMINAL SESSION:');
      PUT SKIP LIST('ENTER: "."<CR> AFTER "V>"');
      PUT SKIP LIST('-----------------------------------');
      PUT SKIP(1);
      TXBUFF(1)=2;
      TXBUFF(2)=7;
      TXBUFF(3)=1;
      TXBUFF(4)=0;
      TXBUFF(5)=7;
      TXBUFF(6)=127;
      TXBUFF(7)=0;
      TXBUFF(8)=0;
      COUNT7C=1;
      PUT SKIP LIST('V>');
      DO WHILE (COUNT7C=1);
         CALL SENDATA;
         PUT SKIP LIST('V>');
         IF VTERM=0 THEN /*END TERMINAL SESSION*/
            DO;
              PUT SKIP LIST('**** END TERMINAL SESSION ****');
              COUNT7C=2;
            END;
         ELSE
            DO;
              CALL INIT;
              CALL RECEIVE;
              PUT LIST('^E^E^HV>');
            END;
      END; /* DO LOOP */
    END;
ELSE
IF SELECT='4' THEN
    DO;
      PUT SKIP LIST('*** VAX COMMAND INSTRUCTIONS ***');
      PUT SKIP LIST('-----------------------------------');
      PUT SKIP LIST('TO DOWNLOAD A FILE FROM THE VAX:');
      PUT SKIP LIST('ENTER THE MESSAGE:');
      PUT SKIP LIST('" !FNAME(VAX).FTYPE(VAX)/XXX` "');
      PUT SKIP LIST('WHERE "XXX"="EXE" FOR NON-TEXT FILES');
      PUT SKIP LIST('AND   "XXX"="TXT" FOR TEXT FILES');
      PUT SKIP LIST('FILE WILL THEN BE IMMEDIATELY SENT');
      PUT SKIP LIST('TO THIS HOST.');
      PUT SKIP LIST('-----------------------------------');
      PUT SKIP LIST('TO UPLOAD A FILE TO THE VAX:');
```

```
        PUT SKIP LIST('1.) ENTER THE MESSAGE:');
        PUT SKIP LIST('" @FNAME(VAX).FTYPE(VAX)/XXX` "');
        PUT SKIP LIST('TO OPEN A VAX FILE BY THE ABOVE NAME');
        PUT SKIP LIST('2.) THEN:');
        PUT SKIP LIST('SEND THE FILE TO THE VAX ADDRESS USING');
        PUT SKIP LIST('THE NORMAL FILE SENDING SELECTIONS.');
        PUT SKIP LIST('------------------------------------');
        PUT SKIP(1);
        TRMODE=1; /*SET VAX CMD MODE FLAG TO TRUE*/
        FRSIZE=128;
        TXBUFF(1)=2;
        TXBUFF(2)=7;
        TXBUFF(3)=1;
        TXBUFF(4)=0;
        TXBUFF(5)=7;
        TXBUFF(6)=127;
        TXBUFF(7)=0;
        TXBUFF(8)=0;
        CALL SENDATA;
        CALL INIT;
        RXBUFF(17)=255;
        CALL RECEIVE;
      END;
   ELSE
   IF SELECT='5' THEN
      COUNT7=2;
   ELSE
   PUT SKIP LIST('INCORRECT OPMODE SELECTION-REENTER:');
END; /* DO LOOP */
PUT SKIP LIST('DISCONNECTING FROM NET-RETURNING TO CP/M.');

TRANS2:

PROCEDURE;

DECLARE
        /* LOCAL VARIABLES */
        COUNT6   FIXED BINARY(7),
        COUNT6A  FIXED BINARY(7),
        COUNT6B  FIXED BINARY(7),
        COUNT6C  FIXED BINARY(7),
        SENDTYPE CHARACTER(1),
        FTYP     CHARACTER(1),
        DRNO     CHARACTER(1),
        /* FILE DATA ENTRY DCLS */
        I FIXED,
        FN CHARACTER(20),
        LOWER CHARACTER(26) STATIC INITIAL
        ('abcdefghijklmnopqrstuvwxyz'),
        UPPER CHARACTER(26) STATIC INITIAL
        ('ABCDEFGHIJKLMNOPQRSTUVWXYZ'),
        /* GLOBAL VARIABLES */
        FILTYP   FIXED BINARY (7) EXTERNAL,
        FNOP     FIXED BINARY (7) EXTERNAL,
        /* GLOBAL DATA STRUCTURES */
```

```
        TXBUFF(1508) FIXED BINARY(7) EXTERNAL,
        1 TXFCB EXTERNAL,
          2 DISK FIXED BINARY(7),
          2 FNAME CHARACTER(8),
          2 FTYPE CHARACTER(3),
          2 TFCB(24) FIXED BINARY(7),
        /* EXTERNAL MODULES */
        SENDATA   ENTRY;

COUNT6 =1;
DO WHILE(COUNT6=1);
   PUT SKIP LIST('TRANSMISSION OPTIONS:');
   PUT SKIP LIST('SEND A MESSAGE    = 1');
   PUT SKIP LIST('SEND A DISK FILE = 2');
   PUT SKIP LIST('*************************************');
   PUT SKIP LIST('ENTER SELECTION ==>');
   GET LIST(SENDTYPE);
   PUT SKIP(2);
   TXBUFF(8)=0;
   IF SENDTYPE='1' THEN
      DO;
        TXBUFF(7)=0;
        CALL SENDATA;
        COUNT6=2;
      END;
   ELSE
   IF SENDTYPE='2' THEN
      DO;
        TXBUFF(7)=16;
        COUNT6A=1;
        DO WHILE(COUNT6A=1);
           PUT SKIP LIST('NATURE OF FILE TO SEND:');
           PUT SKIP LIST('TEXT (ASCII) FILE        = 1');
           PUT SKIP LIST('MACHINE CODE (COM) FILE = 2');
           PUT SKIP LIST('*******************************');
           PUT SKIP LIST('ENTER TYPE OF FILE CHOICE ==>');
           GET LIST(FTYP);
           PUT SKIP(2);
           IF FTYP='1' THEN
              DO;
                FILTYP=1;
                COUNT6A=2;
              END;
           ELSE
           IF FTYP='2'  THEN
              DO;
                FILTYP=2;
                COUNT6A=2;
              END;
           ELSE
           PUT SKIP LIST('INCORRECT CHOICE-REENTER:');
        END;/* DO LOOP */
        COUNT6B=1;
        DO WHILE(COUNT6B=1);
           COUNT6C=1;
```

```
            DO WHILE(COUNT6C=1);
                 PUT SKIP LIST('SPECIFY FILE TO SEND:');
                 PUT SKIP LIST('FILE LOCATED ON:');
                 PUT SKIP LIST('  DRIVE A = 1');
                 PUT SKIP LIST('  DRIVE B = 2');
                 PUT SKIP LIST('**********************');
                 PUT SKIP LIST('ENTER DRIVE NUMBER==>');
                 GET LIST(DRNO);
                 PUT SKIP(2);
                 IF DRNO='1' THEN
                    DO;
                      TXFCB.DISK=1;
                      COUNT6C=2;
                    END;
                 ELSE
                 IF DRNO='2' THEN
                    DO;
                      TXFCB.DISK=2;
                      COUNT6C=2;
                    END;
                 ELSE
                    PUT SKIP LIST('INVALID DRIVE-REENTER:');
            END;/* DO LOOP */
            PUT SKIP LIST('ENTER:"FILENAME.FILETYPE"==>');
            GET LIST(FN);
            PUT SKIP(2);
            FN=TRANSLATE(FN,UPPER,LOWER);
            I=INDEX(FN,'.');
            IF I=0 THEN
               DO;
                 TXFCB.FNAME=FN;
                 TXFCB.FTYPE='   ';
               END;
            ELSE
               DO;
                 TXFCB.FNAME=SUBSTR(FN,1,I-1);
                 TXFCB.FTYPE=SUBSTR(FN,I+1);
               END;
            TXFCB.TFCB(1)=0;
            TXFCB.TFCB(4)=0;
            TXFCB.TFCB(21)=0;
            CALL SENDATA;
            IF FNOP~=1 THEN
               COUNT6B=2;
         END;/* DO LOOP */
         COUNT6=2;
      END;
   ELSE
      PUT SKIP LIST('INCORRECT TRANSMIT MODE-REENTER:');
END; /* DO LOOP */
END TRANS2;

END ETHERNET;/* ISO LAYER 7 MODULE */
```

```
SENDATA: /* PRESENTATION LAYER MODULE-ISO LEVEL 6 */

PROCEDURE;

DECLARE
        /* LOCAL VARIABLES */
        COUNT5A  FIXED BINARY(7),
        VAXMODE  CHARACTER(1),
        DESTADDR CHARACTER(1),
        /* GLOBAL VARIABLES */
        TRMODE   FIXED BINARY(7) EXTERNAL,
        VTERM    FIXED BINARY(7) EXTERNAL,
        FRSIZE   FIXED BINARY(15) EXTERNAL,
        /* GLOBAL DATA STRUCTURES */
        TXBUFF(1508) FIXED BINARY(7) EXTERNAL;

 /*AUTHOR: CAPT. MARK D. STOTZER-USMC-AEGIS GROUP        */
 /*ORIGINAL PROGRAM:07/29/83*/
 /*LAST REVISION: 11/01/83-2200 BY IOANNIS KIDONIEFS*/
 /*                             AND ANTHONY SAKELLAROPOULOS*/
 /*THESIS ADVISOR: PROF. UNO R. KODRES-COMPUTER SCIENCE  */

IF VTERM= 1 THEN /* TERMINAL MODE */
   DO;
     CALL SENDMSG;
     RETURN;
   END;
IF TRMODE= 1 THEN /* VAX COMMAND MODE */
   DO;
     CALL SENDMSG;
     RETURN;
   END;
COUNT5A=1;
DO WHILE(COUNT5A=1);
   PUT SKIP LIST('ADDRESSES ON THIS NETWORK:');
   PUT SKIP LIST('00-03-EA: MDS SYSTEM  = 1');
   PUT SKIP LIST('00-04-0A: MDS SYSTEM  = 2');
   PUT SKIP LIST('00-07-7F: VAX 11/780  = 3');
   PUT SKIP LIST('******************************');
   PUT SKIP LIST('ENTER SELECTION ==>');
   GET LIST(DESTADDR);
   PUT SKIP(2);
   TXBUFF(1)=2;
   TXBUFF(2)=7;
   TXBUFF(3)=1;
   TXBUFF(4)=0;
   IF DESTADDR='1' THEN
      DO;
        TXBUFF(5)=3;
        TXBUFF(6)=234;
        IF TXBUFF(7)=0 THEN
           CALL SENDMSG;
        ELSE
           CALL SENDFILE;
        COUNT5A=2;
```

```
      END;
   ELSE
   IF DESTADDR='2' THEN
      DO;
        TXBUFF(5)=4;
        TXBUFF(6)=10;
        IF TXBUFF(7)=0 THEN
           CALL SENDMSG;
        ELSE
           CALL SENDFILE;
        COUNT5A=2;
      END;
   ELSE
   IF DESTADDR='3' THEN
      DO;
        TXBUFF(5)=7;
        TXBUFF(6)=127;
        TRMODE=0;
        IF TXBUFF(7)=0 THEN
           CALL SENDMSG;
        ELSE
           CALL SENDFILE;
        COUNT5A=2;
      END;
   ELSE
   PUT SKIP LIST('INVALID NET ADDRESS SELECTED-REENTER:');
END; /* DO LOOP */

SENDMSG: /* MESSAGE SENDING MODULE */

PROCEDURE;

DECLARE  /* LOCAL VARIABLES */
         /* GLOBAL VARIABLES */
         FRSIZE  FIXED BINARY(15) EXTERNAL,
         TRMODE  FIXED BINARY(7)  EXTERNAL,
         VTERM   FIXED BINARY(7)  EXTERNAL,
         /* GLOBAL DATA STRUCTURES */
         TXBUFF(1508) FIXED BINARY(7) EXTERNAL,
         RXBUFF(1522) FIXED BINARY(7) EXTERNAL,
         /* EXTERNAL MODULES */
         FILBUF   ENTRY,
         SENDFRAM ENTRY;

IF VTERM=1 THEN /* VIRTUAL TERMINAL MODE */
   DO;
     CALL FILBUF;
     TXBUFF(8)=0;
     IF TXBUFF(9)=96 THEN
        RETURN;
     IF TXBUFF(9)=46 & TXBUFF(10)=96 THEN /*END SESSION*/
        VTERM=0; /*END TERMINAL SESSION*/
     ELSE
        TXBUFF(8)=0;
        CALL SENDFRAM;
```

```
   END;
ELSE
   DO;
     PUT SKIP LIST('MESSAGE SENDER:');
     PUT SKIP LIST('MAXIMUM NUMBER OF CHARACTERS= ');
     PUT LIST(FRSIZE);
     PUT SKIP LIST('ENTER MESSAGE AFTER PROMPT: >');
     PUT SKIP LIST('END MESSAGE WITH ACCENT: ` ');
     PUT SKIP LIST('>');
     CALL FILBUF; /*FILL TRANSMIT BUFFER FROM CONSOLE*/
     CALL SENDFRAM; /* SEND THE MESSAGE */
   END;
END SENDMSG;

SENDFILE: /* FILE SENDING MODULE*/

PROCEDURE;
DECLARE    /* LOCAL VARIABLES */
          COUNT4  FIXED BINARY(7),
          /* GLOBAL VARIABLES */
          FILTYP FIXED BINARY(7) EXTERNAL,
          FNOP   FIXED BINARY(7) EXTERNAL,
          LFRM   FIXED BINARY(7) EXTERNAL,
          /* GLOBAL DATA STRUCTURES */
          TXBUFF(1508) FIXED BINARY(7) EXTERNAL,
          /* EXTERNAL MODULES */
          VAXTXT ENTRY,
          TRNDMA ENTRY,
          OPENDF ENTRY,
          RDISK  ENTRY,
          SENDFRAM ENTRY;

TXBUFF(7)=15;
TXBUFF(8)=0;
CALL OPENDF;
IF FNOP=1 THEN /*FILE NOT ON DISK*/
   DO;
     PUT SKIP LIST('FILE NOT ON DISK-REENTER DATA:');
     PUT SKIP(2);
     RETURN;
   END;
IF TXBUFF(6)=127 & FILTYP=1 THEN
      CALL VAXTXT;  /*VAX TEXT FILE FORMAT CONVERTER*/
ELSE
   DO;
     CALL TRNDMA; /* SET DISK DMA ADDRESS*/
     PUT SKIP LIST('******* FILE TRANSFER BEGINS ******');
     PUT SKIP(2);
     COUNT4=1;
     DO WHILE(COUNT4=1);
        CALL RDISK; /*READ A DISK FILE RECORD*/
        IF LFRM¬=1 THEN
           DO;
             CALL SENDFRAM;
             TXBUFF(8)=1;
```

```
            END;
         ELSE
            COUNT4=2;
      END;/* DO LOOP */
      TXBUFF(8)=255;
      CALL SENDFRAM;
      PUT SKIP LIST('***** FILE TRANSFER ENDS *****');
      PUT SKIP(2);
      RETURN;
   END;
END SENDFILE;

END SENDATA; /* ISO LAYER 6 TRANSMIT MODULE */
```

```
RECDATA: /* ISO LAYER 6 RECEIVE MODULE */

PROCEDURE;

DECLARE  /* GLOBAL DATA STRUCTURES */
         RXBUFF(1522) FIXED BINARY(7) EXTERNAL;

 /*LAST REVISION: 11/01/83-2000 BY IOANNIS KIDONIEFS      */
 /*                              AND ANTHONY SAKELLAROPOULOS*/

 /*ORIGINAL PROGRAM:08/17/83                               */
 /*AUTHOR: CAPT MARK D. STOTZER-USMC-AEGIS GROUP           */
 /*THESIS ADVISOR: PROF. UNO R. KODRES-COMPUTER SCIENCE    */

IF RXBUFF(17)= 0 THEN /* MESSAGE FRAME */
   CALL CONMSG;
ELSE
IF RXBUFF(17)= 15 THEN /* FILE FRAME */
   CALL FILER;
ELSE
   PUT SKIP LIST('RECEIVED IMPROPERLY ENCODED FRAME');


CONMSG: /* MESSAGE RECEIPT MODULE */

PROCEDURE;

DECLARE   /* GLOBAL VARIABLES */
          TRMODE  FIXED BINARY(7) EXTERNAL,
          FRSIZE  FIXED BINARY(15) EXTERNAL,
          VTERM   FIXED BINARY(7) EXTERNAL,
          /* GLOBAL DATA STRUCTURES */
          RXBUFF(1522) FIXED BINARY(7) EXTERNAL,
          /* EXTERNAL MODULES */
          TRMSG ENTRY,
          EMTBUF ENTRY;


     IF VTERM~=1 THEN /* NOT IN VIRTUAL TERMINAL MODE*/
        DO;
          PUT SKIP LIST('***** RECEIVED MESSAGE IS:');
          PUT SKIP(2);
        END;
     CALL TRMSG;  /* SEND THE ACK FRAME */
     CALL EMTBUF; /* DUMP THE RECVD FRAME DATA TO CONSOLE */
     IF VTERM~=1 THEN
       DO;
         PUT SKIP(2);
         PUT SKIP LIST('***** END OF MESSAGE TEXT.');
         PUT SKIP(2);
         PUT SKIP LIST('BACK IN WAIT LOOP-ENTER<CR> TO EXIT=>');
         PUT SKIP LIST('****************************************');
         PUT SKIP(2);
       END;
     ELSE
```

```
    IF RXBUFF(18)=15 THEN /* END OF TERMINAL REPLY */
       PUT SKIP LIST('V>');
END CONMSG;


FILER: /* FILE FRAME RECEIPT MODULE*/

PROCEDURE;

DECLARE     /*  GLOBAL VARIABLES */
            TRMODE    FIXED BINARY(7) EXTERNAL,
            RECFIL    FIXED BINARY(7) EXTERNAL,
            VTERM     FIXED BINARY(7) EXTERNAL,
            /* GLOBAL DATA STRUCTURES */
            1 RXFCB EXTERNAL,
              2 DISK FIXED BINARY(7),
              2 FNAME CHARACTER(8),
              2 FTYPE CHARACTER(3),
              2 TFCB(24) FIXED BINARY(7),
            RXBUFF(1522) FIXED BINARY(7) EXTERNAL,
            /* EXTERNAL MODULES */
            RCVDMA ENTRY,
            DELEDF ENTRY,
            MAKEDF ENTRY,
            WRDISK ENTRY,
            TRMSG  ENTRY,
            CLOSDF ENTRY;

CALL RCVDMA;
IF RXBUFF(18)=0 THEN /* FIRST FILE FRAME */
   DO;
     PUT SKIP LIST('******* FILE RECEIPT BEGINS *******');
     PUT SKIP LIST('  OPENING FILE- RECFROM_.NET:');
     PUT SKIP(2);
     RXFCB.FNAME='RECFROM ';
     RXFCB.FTYPE='NET';
     RXFCB.TFCB(1)=0; /*CURRENT EXTENT FIELD*/
     RXFCB.TFCB(4)=0;
     RXFCB.TFCB(21)=0;
     CALL DELEDF; /*DELETE OLD FILE OF THIS FN.FT*/
     CALL MAKEDF; /*CREATE A NEW ONE*/
     CALL WRDISK; /*WRITE FIRST RECORD(128 BYTES) TO DISK*/
     CALL TRMSG;  /* SEND THE FIRST ACK FRAME */
   END;
ELSE
IF RXBUFF(18)=1 THEN /*INTERMEDIATE FILE FRAME*/
   DO;
     CALL WRDISK; /*WRITE NEXT RECORD TO DISK*/
     CALL TRMSG;  /* SEND THE ACK FRAME */
   END;
ELSE
IF RXBUFF(18)=255 THEN /*LAST(DUMMY) FILE FRAME*/
   DO;
     CALL CLOSDF; /*CLOSE THE DISK FILE*/
     PUT SKIP LIST('******* END FILE RECEIPT *******');
```

```
      PUT SKIP LIST('   SEE FILE(S):RECFROM_.NET');
      PUT SKIP(2);
      CALL TRMSG;  /*SEND THE LAST ACK */
      PUT SKIP LIST('             NOTE:');
      PUT SKIP LIST('--------------------------------------');
      PUT SKIP LIST('IF RECEIVED FILE IS A TEXT FILE FROM');
      PUT SKIP LIST('THE VAX THEN REFORMAT USING:');
      PUT SKIP LIST('"PIP FNAME.FTYPE=RECFROM_.NET[D8C]"');
      PUT SKIP LIST('WHERE FNAME.FTYPE IS YOUR CHOICE');
      PUT SKIP LIST('--------------------------------------');
      PUT SKIP(2);
      IF VTERM=1 THEN
         DO;
           PUT SKIP LIST('STILL IN VAX TERMINAL MODE:');
           PUT SKIP LIST('V>');
         END;
      ELSE
         DO;
           PUT SKIP LIST('IN WAIT LOOP-ENTER<CR> TO EXIT');
           PUT SKIP LIST('*******************************');
           PUT SKIP(2);
         END;
    END;
ELSE
   PUT SKIP LIST(' FRAME TYPE FIELD BYTE 2 INVALID CODE');
END FILER;

END RECDATA; /* ISO LAYER 6 RECEIVE MODULE */
```

APPENDIX H

HIGH LEVEL DESIGN OF A VIRTUAL TERMINAL NETWORK

This Appendix might be useful to the person who may undertake the design of a virtual termianal network. It contains a high level design of a network in which several MDS's will act as VAX/VMS virtual terminals.

The multiplexing of Ethernet interface is the backbone of a design like this. Many routines of the software of the present thesis can be used exactly as they are now, assuming that the system will include no more than nine virtual VAX terminals.

The present configuration of the Ethernet Interface Multiplexing requires that the program "Ethermult" which performs this task will run in a VAX terminal. Since this is undesirable in a virtual terminal network, the program which will perform the coordination of the users, should be able to start execution automatically when a message arrives at the NI1010 board. Also it should be able to supervise any user, regardless of his privileges. In other words it should be able to have access to any user's Virtual Memory Space.

A solution which will fulfill those requirements would be the installation of the coordinating program inside the VMS operating system.

The program could use the "Sys$qio" system routine to listen to the NI1010 board. This routine is interrupt-driven and executes prespecified operations when an I/O event occurs. So, the program could "set the ear" of the system and then for reasons of efficiency go to hibernation. The sequence of operations in the program could be as follows :

As soon as a message arrives at the NI1010, an AST wakes the coordinating program up in order that it will undertake normal operation.

The same procedures which are executed on the real VAX terminal could be followed. So, if the first received message is a carriage return, the "Loginout" procedure is called by the coordinating program to interact with the user for identification and authorization. Naturally, routine "Sendmsg" or a similar one will be used to send the name and password requests to the MDS.

If authorization is succesful the user name and the address of the corresponding virtual terminal is put in a table and the number of current users is updated. Then a mailbox for this user should be created. This mailbox will be the input port for the "Loginout" procedure (this is what cannont be achieved currently), and the output port will be a file.

As soon as a command enters a mailbox it is immediatelly executed in the same manner as if this command had been entered from a VAX terminal. This happens because "Loginout" maps the DCL commands to P0 and P1 spaces of the process that it creates.

Messages other than the inital carriage return for each user are queued and distributed to the appropriate mailboxes in a similar way as in "Ethermult".

An "export" routine will pick up the ready answers and send them to the NI1010.

The use of common event flags will be restricted to the denotation of ready answers. There is no need to use a flag to denote the presence of a user in the system, because the program will know that as soon as succesful log in has been achieved.

Since the commands which are entered a mailbox are executed immediatelly the existance of programs like "Usermult" is not required.

Finally, when a received message is the "Logout" command the user to whom this command is addressed is removed from the system, and the user information table, as well as the number of current users is updated. If all users have exited the system, the program "sets the ear" to the NI1010 and goes to hibernation again.

The software for a virtual terminal network, as visualized by the authors of this thesis, is not much different from program "Ethermult". A good understanding of this program and a thorough knowledge of VAX/VMS facilities should make the accomplishment of this task a relatively easy thing to do.

APPENDIX I

```
        program  loger

c       this program creates the detached process 'LOGER' which
c       runs the image 'LOGINOUT.EXE'.

        implicit          integer*4(a-z)
        integer*4         uic/'0069000E'x/,mbx←iosb(2),ichan,
        1                 stsflag/'00000040'x/

        character*6       user(2)/'SAKELL','SAKELL'/
        character*12      name/' SAKELL     '/
        character*32      pass/' SAKELL                         '/
c       character         mine(2)/'sakell      '/
c       1                          'sakell                       '/
        character*6       input(2)



c       create detached process to run the LOGINOUT image,
c       and set as input the file 'INPUT.DAT' and output the terminal.

        status = sys$creprc(pid,'sys$system:loginout','input.dat',
        1                 '←ttbl:','error.dat',,,
        2                 'LOGER',%val(4),'6881294',,)
        if(.not.status) type *,'ooops!  ',status
        if(status) type *,'loginout image executed'

c       execute a 'show system' command to see if the detached
c       process 'LOGER' has been created.

        status=lib$spawn('$show system')
        type *,pid
        end
```

LIST OF REFERENCES

1. Digital Equipment Corporation, VAX/VMS Internals and Data Structures, 1982.

2. Stotzer, M.D., Design and Implementation of Software Protocol in Intel's MDS Using Ethernet Local Area Network, M.S.Thesis, Naval Postgraduate School, September 1983.

3. Netniyom, T.P., Design and Implementation of Software Protocol in VAX/VMS Using Ethernet Local Area Network, M.S.Thesis, Naval Postgraduate School, June 1983.

4. INTERLAN, Inc., NI3010 UNIBUS Ethernet Communications Controller User Manual (UM-NI3010A), 1982.

5. INTERLAN, Inc., NI1010 UNIBUS Ethernet Communications Controller User Manual (UM-NI1010A), 1982

6. Digital Equipment Corporation, VAX/VMS System Services Reference Manual, 1982.

7. Digital Equipment Corporation, VAX/VMS System Services, I/O Reference Manual, 1982.

8. Digital Equipment Corporation, Programming VMS in VAX-11 FORTRAN/MACRO, 1980.

9. Digital Equipment Corporation, VAX/VMS Real Time User's Guide, 1982.

10. Digital Equipment Corporation, VAX-11 FORTRAN Reference Manual and User's Guide, 1982.

11. Digital Equipment Corporation, VAX/VMS System Services, I/O User's Guide, 1982.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943	2
3.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93943	2
4.	Computer Technology Curricular Office Code 37 Naval Postgraduate School Monterey, California 93943	1
5.	LCDR R. W. Modes, Code 52Mf Department of Computer Science Naval Postgraduate School Monterey, California 93943	1
6.	Professor Uno R. Kodres, Code 52Kr Department of Computer Science Naval Postgraduate School Monterey, California 93943	2
7.	Daniel Green (Code N20E) Naval Surface Warfare Center Dahlgren, Virginia 22449	1
8.	Lieutenant Ioannis Kidoniefs Hellenic Navy General Staff Holargos, Athens - Greece	1
9.	Major Antonios Sakellaropoulos Hellenic Air Force General Staff Holargos, Athens - Greece	1
10.	Dana Small Code 8242 NOSC, San Diego, California 92152	1

11. RCA AEGIS Data Depository 1
RCA Corporation
Government Systems Division
Mail Stop 127-327
Moorestown, New Jersey 22314

207732
Thesis
S15253 Sakellaropoulos
c.1 Multiplexing the
Ethernet interface
among VAX/VMS users.